HPC

HPC BAS4

User's Guide

REFERENCE
86 A2 29ER 09

HPC

HPC BAS4

User's Guide

Hardware and Software

April 2008

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
86 A2 29ER 09

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

Preface

Scope and Objectives

The purpose of this guide is to describe the tools and libraries available as part of the **Bull Advanced Server** (BAS) delivery which allow the development and testing of application programs on the **Bull High Performance Computing (HPC)** clusters. In addition various open source and proprietary tools are described.

Intended Readers

This guide is for users and developers of HPC applications.

Prerequisites

The installation of all hardware and software components of the HPC must have been completed. The HPC administrator must have performed basic administration tasks (creation of users, definition of the file systems, network configuration, etc).

See the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) for more details.

Structure

This guide is organized as follows:

Chapter 1.	*Introduction to the HPC Environment.* Provides a general introduction to Bull's HPC software and hardware environments.

Two types of programming libraries are used when developing programs for the HPC environment: Parallel libraries and Mathematical libraries. These are described in the chapters 2 and 3:

Chapter 2.	*Parallel Libraries.* Describes the Message Passing Interface (MPI) libraries to be used when parallel programming.
Chapter 3.	*Scientific Libraries.* Describes the scientific libraries and scientific functions delivered with the Bull HPC BAS delivery and how these should be invoked. Some of Intel's proprietary libraries are also described.
Chapter 4.	*Compilers.* Describes the compilers available and how to use them.
Chapter 5.	*The User's Environment.* Describes the user's environment on Bull HPC clusters, how the clusters are accessed and the use of the file systems. A description of Modules follows. These can be used to change and compare environments.

Chapter 6.	*Launching an Application.* Different ways of launching cluster resources for an application(s) are explained. The TORQUE batch manager used for batch jobs is described.
Chapter 7.	*Debugging Tools.* Describes Debugging Tools.

For details on system monitoring and application performance optimizations refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER).

Appendix A	A *Troubleshooting guide* which enables you to diagnose some common problems.

Glossary and Acronyms

Provides a Glossary and lists the Acronyms used in the manual.

Bibliography

- Bull HPC BAS4 *Installation and Configuration Guide* (86 A2 28ER)
- Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER)
- Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER)
- Bull HPC BAS4 *Maintenance Guide* (86 A2 46ER)
- The Bull HPC BAS4 *Software Release Bulletin* (SRB) provides release-specific information and details of restrictions resulting from known problems.
- Bull *Voltaire Switches Documentation CD* (86 A2 79ET 01)
- NovaScale 40xx Series and NovaScale 5xxx & 6xxx Series documentation
- NovaScale Master documentation
- Intel® Itanium® 2 *Processor Reference Manual for Software Development and Optimization*

Web Links

http://www.bull.com/novascale/hpc.html

http://www.quadrics.com

http://www.intel.com/design/itanium2/documentation.htm

http://www.linuxhpc.org/

Highlighting

- Commands entered by the user are in a frame in "Courier" font. Example:

```
mkdir /var/lib/newdir
```

- Commands, files, directories and other items whose names are predefined by the system are in "Bold". Example:

The /etc/sysconfig/dump file.

- Text and messages displayed by the system to illustrate explanations are in "Courier New" font. Example:
  `BIOS Intel`
- Text for values to be entered in by the user is in "`Courier New`". Example:
  `COM1`
- *Italics* Identifies referenced publications, chapters, sections, figures, and tables.
- < > identifies parameters to be supplied by the user. Example:
  `<node_name>`

Warning

A Warning notice indicates an action that could cause damage to a program, device, system, or data.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction to the HPC Environment

The term HPC (High Performance Computing) describes the development of large scientific applications and programs, which require a powerful computation facility that can process enormous amounts of data to give highly precise results.

Bull Advanced Server (BAS) is a software suite that is used to operate and manage a Bull HPC cluster. **Bull HPC** clusters are based on Bull NovaScale platforms connected either by a high-speed interconnect **QsNet**^{II} network from **Quadrics** or using **InfiniBand** stacks with **Voltaire** switches or with a **Gigabit Ethernet** network. **BAS** includes both Bull proprietary and Open Source software, which provides the infrastructure for optimal interconnect performance.

The Bull HPC cluster includes an administrative network based on a 10/100 Mbit or a Gigabit Ethernet network, and a separate console management network.

The Bull HPC delivery also provides a full environment for development, including optimized scientific libraries, FORTRAN and C/C++ compilers, MPI libraries, as well as debugging and performance optimization tools.

This manual describes these software components, and explains how to work within the BAS environment.

See the Bull HPC BAS4 *Application Tuning Guide (86 A2 19ER)* for more information on performance optimization tools.

1.1 Software Configuration

BAS is based on a standard Linux distribution, combined with a number of Open Source applications that exploit the best from the Open Systems community. This combined with technology from Bull and its partners, results in a powerful, complete solution for the development, execution, and management of parallel and serial applications simultaneously.

Its key features are:

- Strong manageability, through Bull's systems management suite that is linked to state-of-the-art workload management software.
- High-bandwidth, low-latency interconnect networks.
- Scalable high performance file systems, both distributed and parallel.

All cluster nodes use the same Linux distribution. Parallel commands are provided to supply users and system administrators with single-system attributes, these make it easier to manage and to use cluster resources.

Software installation is carried out by first creating an image on a node, loading this image onto the Management Node, and then distributing it to the other nodes using the **Image Building and Deployment** (KSIS) utility. This distribution is performed via the administration network.

1.2 Program Execution Environment

When a user logs onto the Bull Advanced Server system, the login session is directed to one of several nodes. Upon logging onto the system, the users may then develop and execute their applications. Applications can be executed on other cluster nodes apart from the user login system. For development, the environment consists of:

- Standard Linux tools such as **GCC** (a collection of free compilers that can compile C/C++ and FORTRAN), **GDB Gnu Debugger**, and other third-party tools including the Intel FORTRAN Compiler, the Intel C Compiler and Intel Debugger **IDB**.
- Optimized parallel libraries that are part of the **BAS** software suite. These libraries include the **Bull MPI2** message-passing library. **Bull MPI2** is fully integrated with the **SLURM** resource manager. **Bull MPI2** complies with the MPI1 and 2 standards and is a high performance, high quality native implementation. **Bull MPI2** exploits shared memory for intra-node communication and **MDM** (MPI Data Mover) technology for inter-node communication. It includes a trace and profiling tool, enabling data to be tracked.
- **Modules** software provides a means for predefining and changing environments. Each one includes a compiler, a debugger and library releases which are compatible with each other. So it is easy to invoke one given environment in order to perform tests and then compare the results with other environments.

1.3 Resource Management

For job execution and workload management BAS Software provides an integrated resource management, scheduling and job launch mechanism based on the **Resource Management System (RMS)** from **Quadrics** OR using **SLURM,** an Open Source resource manager.

The resource manager is responsible for the allocation of resources to jobs. The resources are provided by nodes that are designated as compute resources. Processes of the job are assigned to and executed on these allocated resources.

RMS and **SLURM** provide the means to rearrange the cluster into distinct partitions. Serial or parallel jobs may be scheduled for execution within a given partition, provided that the partition has sufficient resources (for example, memory, or number of CPUs) to execute the jobs. The entire system can be designated as a single partition, allowing parallel jobs to run across all of the CPUs of the cluster. Alternatively, the system administrator can divide the system into smaller partitions. See *Chapter 6* for more information.

SLURM provides three key functions. Firstly, it allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for certain period of time so that they can do their work. Secondly, it provides a framework for starting, executing, and monitoring work (typically a parallel job) on a set of allocated nodes. Finally, it arbitrates when there are conflicting requests for resources by managing a queue of pending work.

1.4 Data and Files

Application file I/O operations may be performed using locally mounted storage devices, or alternatively, on remote storage devices using **NFS** or **Lustre** (CFS) file systems for high performance and high availability. By using a separate interconnect for administration and I/O operations, the Bull cluster system administrator is able to isolate user application traffic from administrative operations and monitoring. With this separation, application I/O performance and process communication can be made more predictable while still enabling administrative operations to proceed.

1.5 Exploiting the System

It is essential that users spend time familiarizing themselves with the architecture. The use of resource management tools such as **CPUSET** means that cluster operations can be configured to run in parallel or on separate partitions according to the exigencies of the job. Similarly, wherever possible, you need to optimize your code so that the parallel processing possibilities of the EPIC architecture are fully utilized.

See the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER) for more information on these subjects.

Chapter 2. Parallel Libraries

This chapter describes the following topics:

2.1 Overview of Parallel Libraries

A common approach to parallel programming is to use a message passing library, where a process uses library calls to exchange messages (information) with another process. This message passing allows processes running on multiple processors to cooperate.

Simply stated, a **MPI** (Message Passing Interface) provides a standard for writing message-passing programs. A MPI application is a set of autonomous processes, each one running its own code, and communicating with each other through calls to subroutines of the MPI library.

2.1.1 MPI Versions

Bull provides different MPI libraries for use in the HPC environment.

- The recommended one is **MPI_Bull**. This is the Bull MPI library optimized for the NovaScale architecture. This component is able to run applications in a Quadrics interconnected cluster environment or on a single node. **MPI_Bull** is split into two parts: a global static or dynamic library with which the application is compiled and a dynamic Elan (Quadrics environment) or mono (single node) library which is called when the program is running.
- The second generation MPI library is **MPIBull2**. This library enables dynamic communication with different device libraries; including Quadrics interconnects, InfiniBand **(IB)** interconnects, socket Ethernet/IB/EIB devices or single machine devices.
- Third party MPI libraries are also available. MPICH_Ethernet is provided to allow applications to run in an Ethernet environment instead of the Quadrics interconnect environment. Bull also enables the use of LAM MPI and of Parallel Virtual Machine (PVM) – see sections 2.4.2 and 2.4.3.

Programming with MPI

It is not in the scope of the present guide to describe how to program with MPI. Please, refer to the Web, where you will find complete information. For example, you can refer to the following site: http://www.idris.fr for information in French.

2.1.2 The MPI Data Mover module (MDM)

With a standard MPI library in intra-machine communication, the sender copies data into a shared memory buffer which the receiver then copies into their own memory space. Therefore, two copies are required, as illustrated below. This system of transfer is used by the protocol **short** for messages which are under 32Kbs in size.

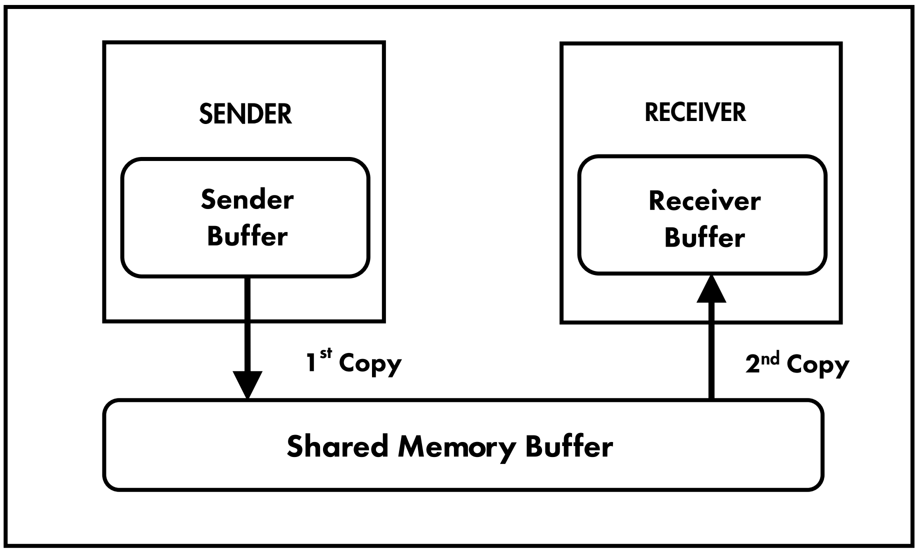


Figure 2-1. MPI Data Mover module – short protocol

With **MPI_Bull** and **MPIBull2**, the **MDM** module enables the use of only one copy, by directly copying the source buffer into the destination one, as illustrated below:

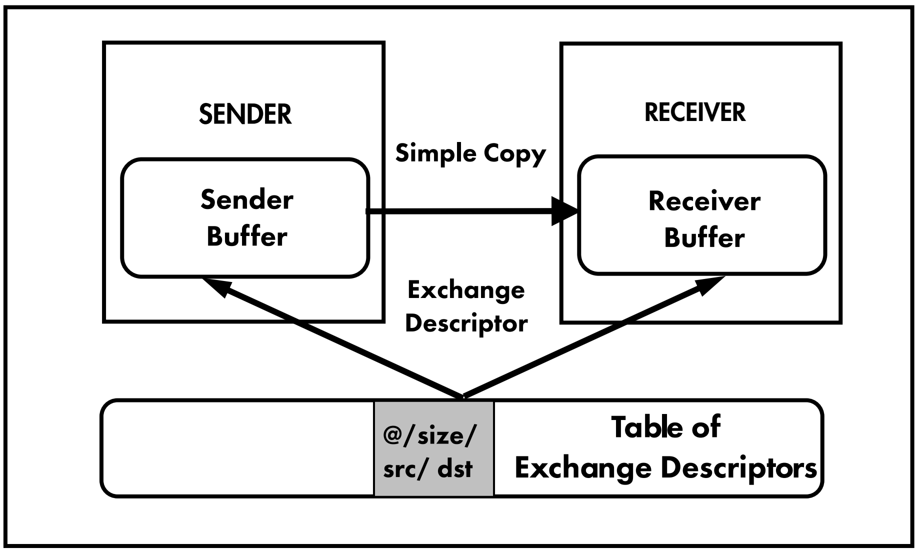


Figure 2-2. MPI Data Mover module – one copy

For messages which are bigger than a determined threshold in size the **MDM** module is used automatically by **MPI_Bull** and **MPIBull2**.

The **MDM** module was created from the **zcopy** module and allows the transmission of long-lasting communications (Memory window allocation) which are used in one-sided communications (MPI-2 reference).

The **MDM** module works in the same way for both **MPI_Bull** version 1.6.x and **MPIBull2** version 2.1.0-x.

2.1.2.1 Using the MDM Module

The **MDM** module being an integral part of **MPI_Bull** and **MPIBull2** has no specific options. However it includes a trace and profiling tool, enabling data to be tracked.

Information related to profiling is in **/proc/mdm/profiler**.

To display the profile, run:

```
$ cat /proc/mdm/profiler
mdm profiling data ======================
```

The trace tool is useful when an application behaves abnormally. To view the events that occurred on different processors, just consult **/proc/mdm/trace/<n>**, where **n** is the CPU number. For example:

```
$ cat /proc/mdm/trace/CPU0
ITC     UID   DESCRIPTION
==================================================
```

You may also watch, in real time, the events occurring for the process whose rank is r and for the application whose MPI jobkey is p, use **/proc/mdm/<p>/trace/rank_<r>**.

```
$ cat /proc/mdm/145231/trace/rank_0
ITC        UID        DESCRIPTION
======================================
```

The status of an application may be known by reading the file **/proc/mdm/<p>/status**, where **p** is the MPI jobkey.

```
$ cat /proc/mdm/145231/status
```

Also, one-sided communications window descriptors are available under the directory **/proc/mdm/<p>/onesided/**, where **p** is the MPI jobkey. In this directory, you may consult the file **rank_<r>**, where **r** is the process rank.

```
$ cat /proc/mdm/145231/onesided/rank_0
ID       WIN                BADDR              SIZE
================================================================
```

To display the **MDM** module release number, run:

```
$ cat /proc/mdm/version
MDM module release mdm (mdm:aravis) 1.0.0-0 [ version kunlock dynamic
32cpus pt2pt profiler ] {mpi_bull >= 1.0.0}
```

2.2 MPI_Bull 1.6.x

Important
MPI_Bull is not supported on **InfiniBand** software stacks.

Bull has perfected a MPI library, called **MPI_Bull**, enabling the exchange of messages between processes in a distributed environment. This model of communication is used by parallel applications.

The **MPI_Bull** library conforms to the MPI -1 standard (refer to *MPI: A Message-Passing Interface Standard*, dated May 5, 1994). http://www-unix.mcs.anl.gov/mpi

The **MPI_Bull** library is optimized for NovaScale hardware architectures. Quadrics Elan3 or Elan4 ensure hardware interconnection.

The BAS4 **MPI_Bull** libraries have been compiled with compilers which correspond to the compilers referred to in chapter 4. This ensures compiler compatibility for any application which uses the BAS4 **MPI_Bull** libraries and these Intel compilers.

MPI_Bull has been developed from the **MPICH** 1.2.6 Open Source library to which several improvements have been made:

- **Mapping of processes on processors**, in order to inhibit scheduler actions (the ideal operation is one process per processor).
- **Introduction of the Futex mechanism** (Fast User mode muTEX), for locks management in the interface.
- **Improvement of MPI_Barrier algorithm** (processes synchronization).
- **Optimization of collective operations** (broadcast, for example).
- **Allocation of a memory area** to benefit from the advantages of the NUMA architecture of NovaScale 5160.
- **Optimization of data copies** between processes located on the same machine, thanks to the MDM module (MPI Data Mover module) for Linux kernel 2.6 and higher.
- **Cpuset** use to locate processes
- **Thread Safety: MPI_Bull** is thread safe (MPI-2 functionality).
- **Introduction of One-Sided communications** which is also a MPI-2 type of functionality.
- **Introduction of a MPI_Bull profiler called Profilecomm** to allow the profiling of applications using MPI_Bull.
- **Integration of Quadrics** Elan library.
- **Integration of mono library** to allow the running of applications on a single node.

Note:

The following restriction applies to the MPI I/O library delivered with **MPI_BULL**.

All nonblocking MPI I/O functions use an **MPIO_Request** object instead of the usual **MPI_Request** object. Accordingly, two functions, **MPIO_Test** and **MPIO_Wait**, are provided to test and wait on these **MPIO_Request** objects. These have the same semantics as **MPI_Test** and **MPI_Wait**, as shown below:

```
int MPIO_Test(MPIO_Request *request, int *flag, MPI_Status*status)
int MPIO_Wait(MPIO_Request *request, MPI_Status *status)
```

The usual functions, for example, **MPI_Test**, **MPI_Wait**, **MPI_Testany**, and so forth, will not work for nonblocking I/O.

This restriction does not exist for the **MPIBULL2** library.

Note:

For more information about **MPICH 1.2.6** Open Source library, please refer to: http://www-unix.mcs.anl.gov/mpi/mpich/

2.2.1 MPI_Bull environments

MPI_Bull can work in two different environments: Quadrics interconnect or single node. For both environments the compilation mode is the same. No specific configuration is required. You simply have to compile your application (appli.exe for example) using the script **mpicc** (C), **mpiCC** (C++), **mpif77** or **mpif90**.

Example: the following command compiles the **appli.c** code using the MPI library:

```
$ mpicc -o /appli.exe /appli.c
```

To produce a dynamically link object, you must set the environment variable LD_LIBRARY_PATH as follows:

```
$ export LD_LIBRARY_PATH=/usr/lib/mpishared:$LD_LIBRARY_PATH
```

Then compile the **appli.c** code using the **–shlib** option:

```
$ mpicc -o shlib /appli.exe /appli.c
```

What follows explains how to launch the application either using **Quadrics** interconnects or in a single node environment.

2.2.2 MPI_Bull and the Quadrics Interconnect cluster environment

A parallel application which uses **MPI_Bull** Message Passing Interface is launched with the **RMS prun** supplied by Quadrics.

Example: the following command launches the **appli.exe** application on 3 processes (**-n** flag), on the 2 first nodes (**-N** flag) of the 'partition' RMS partition (**-p** flag):

```
$ prun -n 3 -N 2 -p partition ./appli.exe
```

For more details about the **RMS prun** command and the options available, run:

```
  $ prun -h.
```

For more information about **RMS** use and configuration, please refer to the Quadrics documentation. For more information about launching an application refer to chapter 6 in this manual.

2.2.3 Bull Mono libraries for the Single Node Environment

In the single node environment, used for testing a program for example, a parallel application may be launched with the **mprun** command which is included as part of the **Bull Mono Libraries**. These libraries are included in the Bull delivery which is provided for Symmetric Multi Processing on a single node.

mprun provides the following options:

-tv	Enable **Totalview** ™ Debugger support
-O	Allow resources to be over-committed. Set this flag to run more than one process per CPU.
-I	Allocate CPUs immediately or fail. By default, **mprun** holds until resources become available.
-C	Use cyclic cpusets.
-n <nprocs>	Specifies the number <nprocs> of MPI processes to start.

For example, the following command launches the **appli.exe** application on 4 processes (**-n** flag).

```
$ mprun -n 4 ./appli.exe
```

-c **<cpus>**	Specify the number of CPUs required per process (default 1).
-M **<rank>**	Bind the process of rank <rank> to a master cpuset rather than to a given CPU. Mainly used in master/slave programs.

-Y <type>	Specify the MPI busy wait strategy: Allowed values are **s** for sched_yield(), **l** for select() and **n** for none.
-i <mode>	Specify how standard input is redirected. See **mprun** man page for more details.
-o <mode>	Specify how standard output is redirected. See **mprun** man page for more details.
-e <mode>	Specify how standard error is redirected. See **mprun** man page for more details.

By default, when running a parallel program, **mprun** forwards standard input to process **0**.

2.3 MPIBull2

MPIBull2 conforms to the MPI-2 standard.

2.3.1 Quick Start for MPIBull2

MPIBULL2 is usually installed in the **/opt/mpi/mpibull2-<version>** directory. The environmental variables **MPI_HOME, PATH, LD_LIBRARY_PATH, MAN_PATH, PYTHON_PATH** will need to be set or updated. These variables should not be set by the user. Use the **setenv_mpibull2.{sh,csh}** environment setting file, which may be sourced from the ${mpibull2_**install_path}/share** directory by a user or added to the profile for all users by the administrator.

2.3.2 MPIBull2 Compilers

The MPIBull2 library has been compiled with the latest Intel compilers, which, according to Bull's test farms, are the fastest ones available for the IA64 architecture. Bull uses Intel **Icc** and **Ifort** compilers to compile the MPI libraries. It is possible for the user to use their own compilers to compile their applications for example **gcc**, however see below.

In order to check the configuration and the compilers used to compile the MPI libraries look at the **${mpibull2_install_path}/share/doc/compilers_version** text file.

MPI applications should be compiled using the MPIBull2 MPI wrapper to compilers:

C programs:	mpicc your-code.c	
C++ programs:	mpiCC your-code.cc	
	or	
	mpic++ your-code.cc	(for case-insensitive file systems)
F77 programs:	mpif77 your-code.f	
F90 programs:	mpif90 your-code.f90	

Wrappers to compilers simply add various command line flags and invoke a back-end compiler; they are not compilers in themselves.

The command below is used to override the compiler type used by the wrapper. **–cc**, **-fc** -, and **cxx** and used for C, Fortran and C++ wrappers.

```
mpi_user >>> mpicc -cc=gcc prog.c -o prog
```

2.3.3 Configuring MPIBull2

MPIBull2 may be used for different architectures including standalone **SMPs**, **Ethernet**, **Infiniband** or **Quadrics** Clusters.

You have to select the device that will use **MPIBull2** before launching an application with MPIBull2.

The list of possible devices available is as follows:

- **osock** is the default device. This uses sockets to communicate and is the device of choice for **Ethernet** clusters.
- **oshm** should be used on a standalone machines, communication is through shared memory.
- **ibmr_gen2**, otherwise known as **InfiniBand multi-rail gen2**. This works over **InfiniBand**'s verbs interface.
- **elanbull2** works with the **Quadrics**' libelan interface.

The device is selected by using the **mpibull2-devices** command with the **–d** switch, for example, enter the command below to use the shared memory device:

```
mpi_user >>> mpibull2-devices -d=oshm
```

For more information on the mpibull2-devices command, see section 2.3.7.

2.3.4 Running MPIBull2

The MPI application requires a launching system in order to spawn the processes onto the cluster. **Bull** provides the **SLURM** Resource Manager as well as the **MPD** subsystem.

For **MPIBull2** to communicate with **SLURM** and **MPD**, the **PMI** interface has to be defined. By default, **MPIBull2** is linked with **MPD**'s **PMI** interface.

If you are using **SLURM**, you must ensure that **MPIBULL2_PRELIBS** includes **-lpmi** so that your **MPI** application can be linked with **SLURM**'s **PMI** library.

For more information on **SLURM**, see chapter 6.

For more information on **MPD**, see section 2.3.6.3

2.3.5 MPIBull2_1.2.x features

MPIBull2_1.2.x includes the following features:

- It only has to be compiled once, supports the NovaScale architecture and is compatible with the more powerful interconnects.
- It is designed so that both development and testing times are reduced and it delivers high performance on **NovaScale** architectures
- Fully compatible with **MPICH2 MPI** libraries. Just set the library path to get all the **MPIBull2** features
- Supports both MPI 1.2 and MPI 2 standard functionalities including
  - Dynamic processes (**osock** only)
  - One-sided communications
  - Extended collectives
  - Thread safety (see the *Thread-Safety* Section below)
  - **ROMIO** including the latest patches developed by Bull

- Multi-device functionality: delivers high performance with an accelerated multi-device support layer for fast interconnects. The library supports:
  - Sockets-based messaging (for **Ethernet**, **SDP**, **SCI** and **EIP**)
  - Hybrid shared memory-based messaging for shared memory
  - InfiniBand architecture multirails driver Gen2
- Easy Runtime Selection: makes it easy and cost-effective to support multiple platforms. With MPIBull2 Library, both users and developers can select drivers at runtime easily, without modifying the application code. The application is built once and works for all interconnects supported by Bull.
- Ensures that the applications achieve a high performance with a high degree of interoperability with standard tools and architectures.
- Common feature for all devices:
  - **FUTEX** (Fast User mode muTEX) mechanism in user mode

2.3.6 Advanced features

2.3.6.1 MPIBull2 Linking Strategies

Designed to reduce development and testing time, **MPIBull2** includes two linking strategies for users.

Firstly, the user can choose to build his application and link dynamically, leaving the choice of the **MPI** driver until later, according to which resources are available. For instance, if a small **Ethernet** cluster is the only resource available, the user compiles and links dynamically, using an **osock** driver, whilst waiting for access to a bigger cluster via a different **InfiniBand** interconnect and which uses the **ibmr_gen2** driver at runtime. For **Quadrics** clusters the **elanbull2** driver is used.

Secondly, the User might want to use an out-of-the-box application, designed for a specified **MPI** device. Bull provides the combination of a **MPI** Core and all its supported devices, which enables the static libraries to be linked by the User's application.

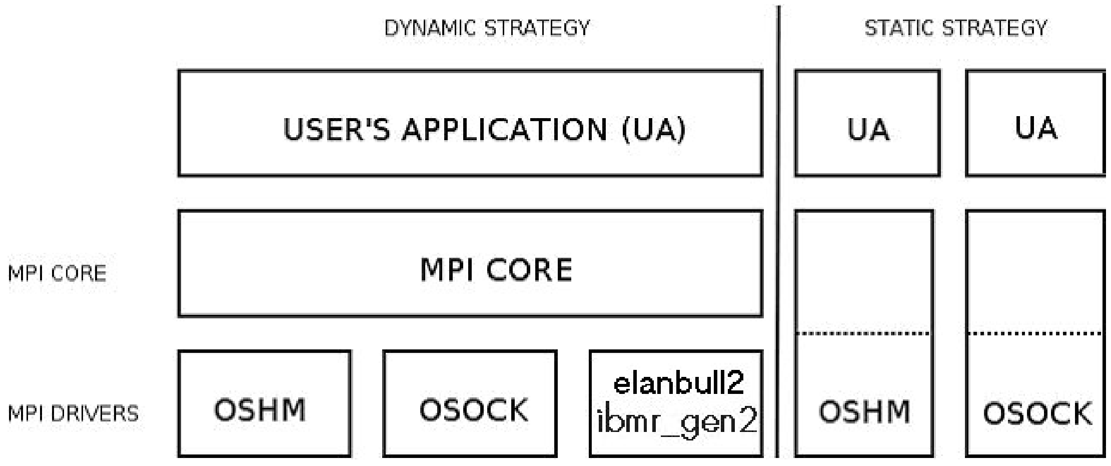


Figure 2-3. MPIBull2 Linking Strategies

2.3.6.2 Thread-safety

If the application needs an **MPI** Library which provides **MPI_THREAD_MULTIPLE** thread-safety level, then choose a device which supports **thread safety** and select a ***_ts device.** Use the **mpibull2-device** commands.

Note:

Thread-safety within the MPI Library requires data locking. Linking with such a library may impact performance. A loss of around 10 to 30% has been observed on microbenchmarks

Not all MPI Drivers are delivered with a thread-safe version. Devices known to support **MPI_THREAD_MULTIPLE** include **osock, oshm** and **elanbull2**.

2.3.6.3 Using MPD

MPD is a simple launching system from **MPICH-2**.

To use it, you need to launch the **MPD** daemons on Compute hosts.

If you have a single machine, just launch **mpd &** and your **MPD** setup is complete.

If you need to spawn **MPI** processes across several machines, you must use **mpdboot** to create a launching ring on the cluster. This is as follows:

Create the hosts list:

```
mpi_user >>> export cluster_machines="host1 host2 host3 host4"
```

- Create the file used to store host information:

```
mpi_user >>> for i in $cluster_machines; do echo "$i" >> machinefiles; done
```

- Boot the MPD system on all the hosts:

```
mpi_user >>> mpdboot -n $(cat $clustermachines | wc -l) -f machinefiles
```

- Check if everything is OK:

```
mpi_user >>> mpdtrace
```

- Run the application or try hostname:

```
mpi_user >>> mpiexec -n 4 ./your_application
```

MPI Process Daemons (MPD) run on all nodes in a ring like structure and may be used in order to manage the launch of the different processes. **MPIBull2** library is **PMI** compliant which means it can interact with any other **PMI PM**. This software has been developed by **ANL**. In order to set up the system the **MPD** ring must firstly be knitted, by following the procedure below:

- At the $HOME prompt edit the **.mpd.conf** file by adding something like **MPD_SECRETWORD=your_password** and `chmod 600` to the file.

- Create a boot sequence file. Any type of file may be used. The **MPD** system will by default use the **mpd.hosts** file in your **$HOME** directory if a specific file is not specified in the boot sequence. This contains a list of hosts, separated by carriage returns. Semi-colons can be added to the host to specify the number of CPUS for the host, for example.

```
host1:4
host2:8
```

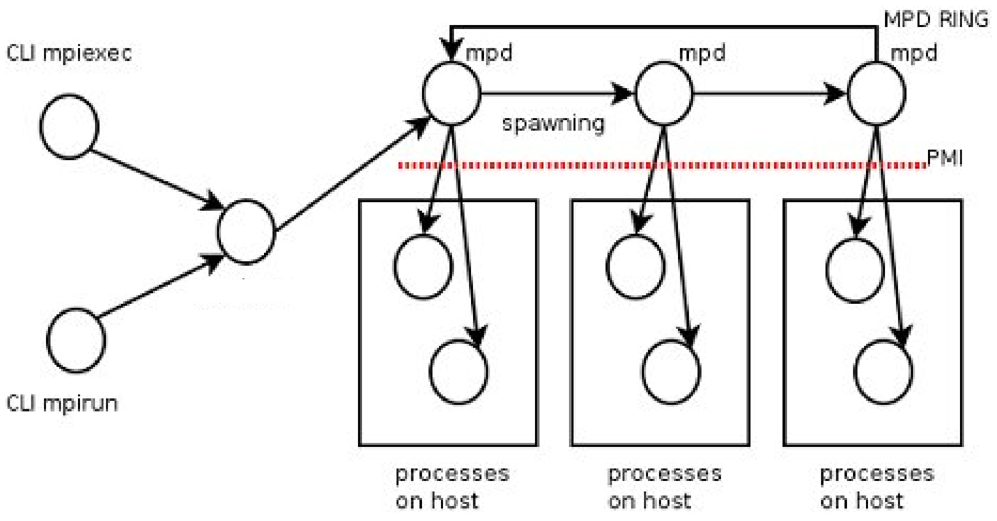


Figure 2-4. MPD ring

- Boot the ring by using the **mpdboot** command, and specify the number of hosts to be included in the ring.

```
mpdboot -n 2 -f myhosts_file
```

Check that the ring is functioning correctly by using the **mpdtrace** or **mpdringtest** commands. If everything is okay, then jobs may be run on the cluster.

2.3.7 MPIBull2 Tools

2.3.7.1 MPIBull2-devices

This tool may be used to change the user's preferences. It can also be used to disable a library. For example, if the program has already been compiled and the intention is to use dynamic MPI Devices which have already been linked with the MPI Core, then it is now possible to specify a particular runtime device with this tool. The following options are available with **MPIBULL2-devices**:

-dl	Provides list of drivers. This is also supported by MPI wrappers.
-dlv	Provides list of drivers with versions of the drivers.

```
mpi_user >>> mpibull2-devices -dl
MPIBULL2 Communication Devices :
+ Original Devices :
*oshm       : Shared Memory device, to be used on a single machine
[static][dynamic]
*osock      : Socket protocol (can be used over IPoIB, SDP, SCI...)
[static][dynamic]
******
```

-c Obtains details of the user's configuration.

```
mpi_user >>> mpibull2-devices -c
MPIBULL2 home : /install_path
User prefs    :
  \__ Directory                   : /home_nfs/mpi_user/.MPIBull2/
  \__ Custom devices              : /home_nfs/mpi_user/.MPIBull2//site_libs
  \__ MPI Core flavor             : Standard / Error detection on
  \__ MPI Communication Driver    : oshm (Shared Memory device, to be used on a
single machine) [static][dynamic]
```

-d=xxx Sets the specified communication device driver.

```
mpi_user >>> mpibull2-devices -d=ibmr_gen2
```

2.3.7.2 mpibull2-launch

This is a meta-launcher which connects to whatever process manager is specified by the user. It is used to ensure compatibility between different process manager launchers, and also to allow users to specify their custom key bindings.

The purpose of **mpibull2-launch** is to help users to retain their launching commands **Mpibull2-launch** also interprets user's special keybindings, in order to allow the user to retain their preferences, regardless of the cluster and the **MPI** library. This means that the user's scripts will not need changing, except for those environment variables which are required.

The **mpibull2-launch** tool provides default keybindings. The user can check them using the **--metahelp** option. If the user wishes to check some of the **CPM** (Cluster Process Manager) special commands, they should use **--options** with the **CPM** launch name command (e.g. **--options srun**).

Some tool commands and 'device' functionalities rely on the implementation of the **MPI** components. This simple tool maps keybindings to the underlying **CPM**. Therefore, a unique command can be used to launch a job on a different CPM, using the same syntax. **mpibull2-launch** system takes in account the fact that a user might want to choose their own keybindings. A template file, named **keylayout.tmp1**, may be found in the tools rpm which may be used to construct individual keybinding preferences.

Launching a job on a cluster using mpibull2-launch

For a **SLURM CPM** use a command similar to the one below and set **MPIBULL2_LAUNCHER=srun** to make this command compatible with the **SLURM CPM**.

```
mpibull2-launch -n 16 -N 2 -ptest ./job
```

Example for a user who wants to use the Y key for the partition

```
PM Partition to use+Y:+partition:
```

The user should edit a file using the format found in the example template, and then add custom bindings using the **–custom_keybindings** option. The + sign is used to separate the fields. The first field is the name of the command, the second the short option, with a colon if an argument is needed, and the third field is the long option.

2.3.7.3 mpiexec

This is a launcher which connects to the MPD ring.

2.3.7.4 mpirun

This is a launcher which connects to the MPD ring.

2.3.7.5 mpicc, mpiCC, mpicxx, mpif77 and mpif90

These are all compiler wrappers and are available, for C, C++, Fortran 77 and Fortran 90 languages. These allow the user to concentrate on developing the application without having to think about the internal mechanics of MPI. The man page files provide more details about wrappers.
When using compiling tools, they need to know which communication device and a linking strategy they should use. The compiling tools parse as long as some of the following conditions have been met:

- The device and linking strategy has been specified in the command line using the **-sd** options.
- The environment variables **DEF_MPIDEV, DEF_MPIDEV_LINK** (required to ensure compatibility), **MPIBULL2_COMM_DRIVER**, and **MPIBULL2_LINK_STRATEGY** have been set.
- The preferences have already been set up; the tools will use the device they find in the environment using the **MPIBULL2-devices** tool.
- The tools take the system default, using dynamic socket device.

Note:
One can obtain better performance using the **–fast/-static** options to link statically with one of the dependent libraries using the commands below:

```
mpicc –static prog.c
mpicc –fast prog.c
```

2.3.8 MPIBull2 – Example of use

2.3.8.1 Setting up the devices

When compiling an application the user may wish to keep those makefiles and build files which have already been generated. Bull has taken this into account. The code and build files can be kept as they are. All the user needs to do is to set up a few variables or use the **MPIBULL2-devices** tool.

During the installation process, the **/etc/profile.d/mpibull2.sh** file will have been modified by the System Administrator according to the user's needs. This file determines the default settings (by default the rpm sets the **osock** socket/TCP/IP driver). It is possible to override these settings by using environment variables – this is practical as it avoids modifying makefiles - or by using the tools options. For example, the user can statically link their application against a static driver as shown below. The default linking is dynamic, and this enables drive modification during runtime. Linking statically, as shown below, overrides the user's preferences but does not change them.

```
mpi_user >>> mpicc -sd=ibmr_gen2 prog.c -o prog
mpicc : Linking statically MPI library with device (ibmr_gen2)
```

The following environment variables may also be used

MPIBULL2_COMM_DRIVER	Specifies the default device to be linked against
MPIBULL2_LINK_STRATEGY	Specifies the link strategy (the default is dynamic) (required to ensure compatibility)
MPIBULL2_MPITOOLS_VERBOSE	Provides information when building (the default is verbose off)

```
mpi_user >>> export DEF_MPIDEV=ibmr_gen2
mpi_user >>> export MPIBULL2_MPITOOLS_VERBOSE=1
mpi_user >>> mpicc prog.c -o prog
mpicc : Using environment MPI variable specifications
mpicc : Linking dynamically MPI library with device (ibmr_gen2)
```

2.3.8.2 Submitting a job

If a user wants to submit a job, then according to the process management system, they can use **MPIEXEC**, **MPIRUN**, **SRUN** or **MPIBULL2-LAUNCH** to launch the processes on the cluster (the online man pages gives details of all the options for these launchers)

2.3.9 Debugging

2.3.9.1 Parallel GDB

With the **mpiexec** launching tool it is possible to add the Gnu DeBugger in the global options by using **-gdb**. All the **gdb** outputs are then aggregated, indicating when there are differences between processes. The **-gdb** option is very useful as it helps to pinpoint faulty code very quickly without the need of intervention by external software.

Refer to the **gdb** man page for more details about the options which are available.

2.3.9.2 Totalview

Totalview is a proprietary software application and is not included in the **BAS4** distribution. See chapter 7 for more details.

It is possible to submit jobs using the **SLURM** resource manage with a command similar to the format below or via MPD.

```
totalview srun –a <args> ./prog <progs_args>
```

Alternatively, it is possible to use MPI process daemons (**MPD**) and to synchronize **Totalview** with the processes running on the MPD ring.

```
mpiexec -tv <args> ./prog <progs_args>
```

2.3.9.3 MARMOT MPI Debugger

MARMOT is an **MPI** debugging library. **MARMOT** surveys and automatically checks the correct usage of the MPI calls and their arguments made during runtime. It does not replace classical debuggers, but is used in addition to them.

The usage of the MARMOT library will be specified when linking and building an application. This library will be linked to the application and to the **MPIBULL2** library.

It is possible to specify the usage of this library manually by using the **MPIBULL2_USE_MPI_MARMOT** environment variable, as shown in the example below;

```
export MPIBULL2_USE_MPI_MARMOT=1
mpicc bench.c -o bench
```

or by using the **-marmot** option with the MPI compiler wrapper, as shown below:

```
mpicc -marmot bench.c -o bench
```

See the documentation in the share section of the marmot package, or go to http://www.hlrs.de/organization/amt/projects/marmot/ for more details on Marmot.

2.3.10 Mpibull2-params

mpibull2-params is a tool that is used to list/modify/save/restore the environment variables that are used by the **mpibull2** library and/or by the communication device libraries (**InfiniBand**, **Quadrics**, etc.). The behaviour of the **mpibull2** MPI library may be modified using environment variable parameters to meet the specific needs of an application. The purpose of the **mpibull2-params** tool is to help **mpibull2** users to manage different sets of parameters. For example, different parameter combinations can be tested separately on a given application, in order to find the combination that is best suited to its needs. This is facilitated by the fact that **mpibull2-params** allow parameters to be set/unset dynamically.

Once a specific combination of parameters has been tested and found to be good for a particular context, they can be saved into a file by a mpibull2 user. Using the **mpibull2-params** tool, this file can then be used to restore the set of parameters, combined in exactly the same way, at a later date.

Notes:

- The effectiveness of a set of parameters will vary according to the application. For instance, a particular set of parameters may ensure low latency for an application, but reduce the bandwidth. By carefully defining the parameters for an application the optimum, in terms of both latency and bandwidth, may be obtained.
- Some parameters are located in the **/proc** file system and only super users can modify them.

The entry point of the **mpibull2-params** tool is an internal function of the environment. This function calls an executable to manage the MPI parameter settings and to create two temporary files. According to which shell is being used, one of these two files will be used to set the environment and the two temporary files will then be removed. To update your environment automatically with this function, please source either the **$MPI_HOME/bin/setenv_mpibull2.sh** file or the **$MPI_HOME/bin/setenv_mpibull2.csh** file, according to which shell is used.

2.3.10.1 The mpibull2-params command

SYNOPSIS

```
mpibull2-params <operation_type> [options]
```

Actions

The following actions are possible for **mpibull2-params** command:

-l List the MPI parameters and their values

-f List families of parameters

-m Modify a MPI parameter

-d Display all modified parameters

-s Save the current configuration into a file

-r   Restore a configuration from a file

-h   Show help message and exit

Options

The following options and arguments are possible for the **mpibull2-params** command.

Note:
The options shown can be combined, for example, **-li** or can be listed separately, for example **–l –i**. The different option combinations for each argument are shown below.

-l [iv] [PNAME]
List current default values of all MPI parameters. Use the PNAME argument (this could be a list) to specify a precise MPI parameter name or just a part of a name. Use the **-v** (verbose) option to also display all possible values, including the default. Use the **-i** option to list all information.

Examples

- This will list all the parameters with the string 'all' or 'shm' in their name:

```
mpibull2-params -l all shm
```

- This will display all information - possible values, family, purpose, etc. for each parameter name which includes the string 'all'. This command will also indicate when the current value has been returned by **getenv()** i.e. the parameter has been modified in the current environment:

```
mpibull2-params -li all
```

`mpibull2-params -l | grep -e all -e shm` will return the same result.

- This will display current and possible values for each parameter name which includes the string 'rom'. It is practical to run this command before a parameter is modified:

```
mpibull2-params -lv rom
```

-f [l[iv]] [FNAME]
This will list all the default family names. Use the FNAME argument (this could be a list) to specify a precise family name or just a part of a name. Use the **-l** option to list all parameters for the family specified. **–l**, **-v** and **-i** options are as described above.

Examples

- List all family names with the string 'band' in their names:

```
mpibull2-params -f band
```

- For each family name with the string 'band' inside, list all the parameters and current values.

```
mpibull2-params -fl band
```

-m [v] [PARAMETER VALUE]
Modify a MPI PARAMETER with VALUE. The exact name of the parameter should be used to modify a parameter. The parameter is set in the environment, independently of the shell syntax (ksh/csh) being used. The keyword 'default' should be used to restore the parameter to its original value. If necessary, the parameter can then be unset in its environment. The -m operator lists all the modified MPI parameters by comparing all the MPI parameters with their default values. If none of the MPI parameters have been modified then nothing is displayed. The –m operator is like the -d option. Use the -v option for a verbose mode.

Examples

- This will set the ROMIO_LUSTRE parameter in the current environment.

```
mpibull2-params -m mpibull2_romio_lustre true
```

- This will unset the ROMIO_LUSTRE parameter in the environment in which it is running and returns it to its default value.

```
mpibull2-params -m mpibull2_romio_lustre default
```

-d [v]
This will display the difference between the current and the default configurations. Displays all modified MPI parameters by comparing all MPI parameters with their default values.

-s [v] [FILE]
This will save all modified MPI parameters into FILE. It is not possible to overwrite an existing file, an error will be returned if one exists. Without any specific arguments, this file will create a file named with the date and time of the day in the current directory. This command works silently by default. Use the -v option to list all modified MPI parameters in a standard output.

Example

- This command will, for example, try to save all the MPI parameters into the file named `Thu_May_10_15_50_28_2007`.

```
mpibull2-params -sv
```

Output Example

```
save the current setting :
mpibull2_mpid_xxx=1
1 parameter(s) saved.
```

-r [v] [FILE]
Restore all the MPI parameters found in FILE and set the environment. Without any arguments, this will restore all modified MPI parameters to their default value. This command works silently, in the background, by default. Use the -v option to list all restored parameters in a standard output.

Example

- Restore all modified parameters to default:

```
mpibull2-params -r
```

-h
Displays the help page

2.3.10.2 Family names

The command **mpibull2-params –f** will list the parameter family names which are possible for a particular cluster environment.

Some of the parameter family names which are possible for Bull **BAS4** are listed below.

LK_Ethernet_Core_driver
LK_IPv4_route
LK_IPv4_driver
OpenFabrics_IB_driver
Marmot_Debugging_Library
MPI_Collective_Algorithms
MPI_Errors
CH3_drivers
CH3_drivers_Shared_Memory
Execution_Environment
Infiniband_RDMA_IMBR_mpibull2_driver
Infiniband_Gen2_mpibull2_driver
UDAPL_mpibull2_driver
IBA-VAPI_mpibull2_driver
MPIBull2_Postal_Service
MPIBull2_Romio

Run the command **mpibull2-params <fl> <family>** to see the list of individual parameters that are included in the parameter families used within your cluster environment.

2.4 Third party MPI libraries

2.4.1 MPICH_Ethernet

Bull supplies **MPICH_Ethernet** (version 1.2.6), this is to be used with Ethernet interconnects.

Modify the file **/opt/mpi/mpich_ethernet-1.2.6/share/machines.LINUX** in order to set the host name of the corresponding interface (Administration Network or Dedicated Network) and the number of processors for each machine. For example:

```
ns0:4
ns1:4
ns2:4
ns3:4
```

The program which uses **MPICH_Ethernet** must be compiled using the appropriate wrapping tool, for example **mpicc**, **mpif77**, etc. Launch the program with the following command where **np** is the number of processes, and **appli.exe** is the name of the application using MPI:

```
$mpirun -np 4 ./appli.exe
```

For more details, see the *Installation and User's Guide to MPICH, a portable implementation of MPI* for the device ch_p4 which is available from http://www-unix.mcs.anl.gov/mpi/mpich/

2.4.2 LAM MPI

Bull delivers **LAM version -7.0.6-5** on the Bull Linux AS4 V5.1 DVD. However, this is not installed automatically.

For more information on **LAM/MPI** and to download the latest source files, please refer to www.lam-mpi.org .

2.4.3 Parallel Virtual Machine (PVM)

Bull delivers **PVM version -3.4.4-21** on the Bull Linux AS4 V5.1 DVD. This is installed on Compute and Login Nodes.

For more information on **PVM** and to download the latest source files, please refer to www.csm.ornl.gov/pvm/pvm_home.html

2.5 Managing your MPI environment

Bull provides different MPI libraries for the different requirements of users. In order to help users manage different environment configurations, Bull also ships modules and these can be used to switch from one MPI library environment to another. This relies on the module software – see chapter 5.

The directory used to store the module files is **/opt/mpi/modulefiles/**, into which the different module files that include the **mpich_ethernet, vltmpi** libraries for **Voltaire InfiniBand,** and **MPIBull2** environments are placed.

Important
It is recommended that a file is created, for example **99-mpimodules.sh** and **99-mpimodules.sh .csh,** and this is added to the **/etc/profile.d/** directory. The line below should be pasted into this file. This will make the configuration environment available to all users.

```
module use -a /opt/mpi/modulefiles
```

1. To check the modules which are available run the following command:

```
module av
```

This will give output similar to that below:

```
-------------------------------------------------------------
------------------- /opt/mpi/modulefiles ------------------
mpibull2/1.2.8-1.t      mpich/1.2.7-p1         vltmpi/24-1
-------------------------------------------------------------
```

2. To see which modules are loaded run the command:

```
module li
```

This will give output similar to that below:

```
-------------------------------------------------------------
Currently Loaded Modulefiles:
  1) oscar-modules/1.0.3
-------------------------------------------------------------
```

3. To change MPI environments run the following commands according to your needs:

```
module load mpich
module li
```

```
-------------------------------------------------------------
Currently Loaded Modulefiles:
  1) oscar-modules/1.0.3   2) mpich/1.2.7-p1
-------------------------------------------------------------
```

4. To check which MPI environment is loaded run the command below:

```
which mpicc
```

   This will give output similar to that below:

```
-------------------------------------------------------------
/opt/mpi/mpich-1.2.7-p1/bin/mpicc
-------------------------------------------------------------
```

5. To remove a module (e.g. mpich) run the command below:

```
module rm mpich
```

6. Then load the new MPI environment by running the load command, as below:

```
module load mpibull2
```

2.6 Profiling with mpianalyser

mpianalyser is a profiling tool, developed by Bull for its own **MPI_Bull** implementation. **mpianalyser** includes **profilecom,** a non-intrusive tool that allows the display of data which has been logged from counters when the application runs.

For more information on **profilecomm** and how it should be used refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER)

Chapter 3. Scientific Libraries

This chapter describes the following topics:

Important: See the **BAS4** System Release Bulletin for details of the Scientific Libraries included with your delivery.

3.1 Overview

Scientific Libraries are sets of tested, validated and optimized functions which spare users the need to develop such subprograms themselves.

The advantages of these scientific libraries are:
- Portability
- Support for different types of data (real, complex, double precision, etc.)
- Support for different kinds of storage (banded matrix, symmetrical, etc.)

Delivery

The scientific libraries **BLACS**, **SCALAPACK**, **FFTW**, **Blocksolve95**, **SuperLU**, **PETSC** use **MPI** (Message Passing Interface). They are delivered in different environmental versions according to the implementation to be used. **BAS4** uses the following implementation:

- **MPICH_Ethernet** for clusters using Gigabit Ethernet interconnect

Note:
These require a machinefile. The default is
/usr/mpi/mpich_ethernet<version>/machines.LINUX

This file contains the system names and the number of processors. For example,

- `Bass:4`
- `Molson:8`

Additionally, all of the systems that will be running must have the same library **revisions.**

3.2 Intel Math Kernel Library

This library, which has been optimized by **Intel** for its processors contains, among other things, the following libraries: **BLAS**, **LAPACK** and **FFT**.

The Intel Cluster **MKL** is a fully thread-safe library.

An installation notice is provided by Bull with the library delivery.

The library is located in the **/opt/intel/mkl<release_nb>/** directory.

To use it, the environment has to be set by updating the **LD_LIBRARY_PATH** variable:

```
export LD_LIBRARY_PATH=/opt/intel/mkl<release_nb>/lib/64:$LD_LIBRARY_PATH
```

Example for **MKL** 7.2:

```
export LD_LIBRARY_PATH=/opt/intel/mkl72/lib/64:$LD_LIBRARY_PATH
```

3.3 Intel Cluster Math Kernel Library

The Intel Cluster Math Kernel Library contains all the highly optimized math functions of the Math Kernel Library plus **ScaLAPACK** for Linux Clusters.

The Intel Cluster MKL is a fully thread-safe library and provides C and **Fortran** interfaces.

An installation notice is provided by Bull with the library delivery.

The Cluster MKL library is located in the **/opt/intel/mkl<release_nb>cluster/** directory.

3.4 BLAS

BLAS stands for Basic Linear Algebra Subprograms.

This library contains linear algebraic operations that include matrixes and vectors. Its functions are separated into three parts:

- Level 1 routines to represent vectors and vector/vector operations.
- Level 2 routines to represent matrixes and matrix/vector operations.
- Level 3 routines mainly for matrix/matrix operations.

This library is included in the Intel MKL package.

For more information see www.netlib.org/blas.

3.5 BLACS

BLACS stands for Basic Linear Algebra Communication Subprograms.

BLACS is a specialized communications library (using message passing). After defining a process chart, it exchanges vectors, matrices and blocks and so on. It can be compiled on top of MPI or PVM systems.

BLACS uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs.

More information is available at the following location:
www.netlib.org/blacs/index.html

3.5.1 Using BLACS

There are multiple versions of BLACS. One uses MPICH and one uses MPIBULL2. These libraries are located in the following directories:

```
/opt/scilibs/blacs/blacs_mpich_ethernet-<versions>/
/opt/scilibs/blacs/blacs_mpibull2-<version>
```

The libraries include the following:

```
libblacsCinit_MPI-LINUX-0.a
libblacsF77init_MPI-LINUX-0.a
libblacs_MPI-LINUX-0.a
```

3.5.2 Testing Library Installation

The installation of the library can be tested using the tests found under the following directory:

/opt/scilibs/BLACS/blacs-1.1-p3/tests

First, the **MPI_HOME** and **LD_LIBRARY_PATH** variables must be set up to point to the MPI libraries that are to be tested. The following example uses the MPICH library.

```
:export MPI_HOME=/opt/mpi/mpich_ethernet/
   export PATH=$MPI_HOME/bin:$PATH
   export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
```

Running the Tests

Then, run the tests as follows:

```
mpirun -np 4 xCbtest_MPI-LINUX-0
mpirun -np 4 xFbtest_MPI-LINUX-0
```

3.6 PBLAS

PBLAS stands for Parallel Basic Linear Algebra Subprograms.

PBLAS is the parallelized version of **BLAS** for distributed memory machines. It requires cyclic distribution by matrix block that the BLACS library offers.

This library is included in the Intel **MKL** package.

3.7 LAPACK

LAPACK stands for Linear Algebra PACKage.

This is a set of Fortran 77 routines used to resolve linear algebra problems such as the resolution of linear systems, eigenvalue computations, matrix computations, etc. However, it is not written for a parallel architecture.

This library is included in the Intel **MKL** package.

3.8 SCALAPACK

SCALAPACK stands for: SCAlable Linear Algebra PACKage.

This library is the scalable version of LAPACK. Both libraries use block partitioning to reduce data exchanges between the different memory levels to a minimum. SCALAPACK is above all used for eigenvalue problems and factorizations (LU, Cholesky and QR). Matrices are distributed using BLACS.

More information can be found at the following location:
www.netlib.org/scalapack/index.html

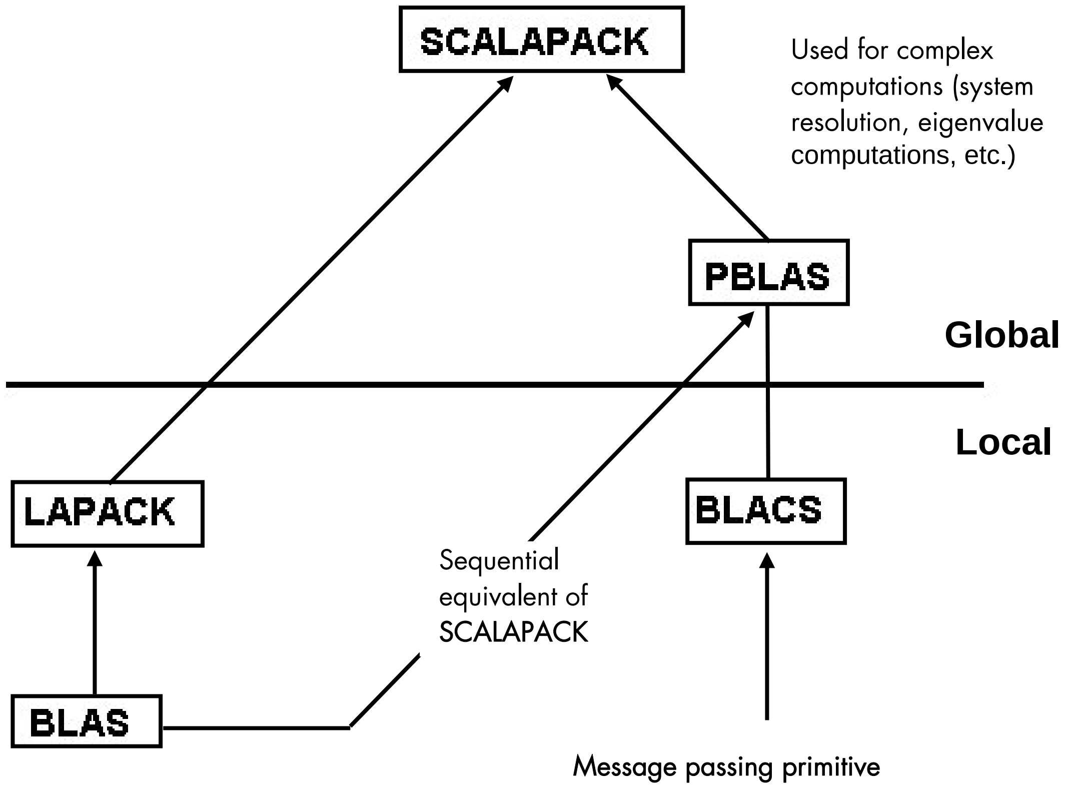

SCALAPACK
Used for complex computations (system resolution, eigenvalue computations, etc.)
PBLAS
Global
Local
LAPACK
BLACS
Sequential equivalent of SCALAPACK
BLAS
Message passing primitive

Figure 3-1. Interdependence of the different mathematical libraries

Local component routines are called by a single process with arguments residing in local memory.
Global component routines are synchronous and parallel. They are called with arguments that are matrices or vectors distributed over all the processes.
SCALAPACK uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs.

The default installation for these two libraries is as follows:

```
/opt/scilibs/scalapack/scalapack_<version>/<mpilib>
```

The following library is provided:

Libscalapack.a

Several tests are provided in the following directory:

/opt/scilibs/scalapack/scalapack_<version>/tests

3.9 Blocksolve95

BlockSolve95 is a scalable parallel software library primarily intended for the solution of sparse linear systems that arise from physical models, especially problems involving multiple degrees of freedom at each node.

BlockSolve95 uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs.

The default installation for these libraries is as follows:

/opt/scilibs/BlockSolve95_<version>/<mpilibrary>/lib/lib0/linux

The following library is provided:

libBS95.a

Some examples are also provided in the following directory.

/opt/scilibs/BlockSolve95_<version>/<mpilibrary>/examples

For more information see. http://www.mcs.anl.gov/sumaa3d/BlockSolve/index.html.

3.10 SuperLU

This library is for the direct solution of large, sparse, nonsymmetrical systems of linear equations on high performance machines. The routines will perform an LU decomposition with partial pivoting and triangular systems solves through forward and back substitution. The factorization routines can handle non-square matrices, but the triangular solves are performed only for square matrices. The matrix commands may be pre-ordered, either though library or user supplied routines. This pre-ordering for sparsely is completely separate from the factorization. Working precision iterative refinement subroutines are provided for improved backward stability. Routines are also provided to equilibrate the system, estimate the condition number, calculate the relative backward error and estimate error bounds for the refined solutions. SuperLU_Dist is for distributed memory.

More information can be found at the following location:

http://crd.lbl.gov/~xiaoye/SuperLU/#superlu_dist

3.10.1 SuperLU Libraries

The following SuperLU Libraries are provided:

/opt/scilibs/SuperLU_DIST/SuperLU_DIST_<version>/<mpilib>/lib/superlu_lnx_ia64.a
/opt/scilibs/SuperLU_SEQ<x>/SuperLU_ SEQ<x>-2.0/lib/superlu_lnx_ia64.a
/opt/scilibs/SuperLU_SMP/SuperLU_ SMP-1.0/lib/superlu_lnx_ia64.a

Test are provided for each library under the following directory:

/opt/scilibs/SuperLU/<versions>/test directory

3.11 FFTW

FFTW stands for Fastest Fourier Transform in the West. FFTW is a C subroutine library for computing a discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and with both real and complex data.

There are three versions of FFTW in this distribution. They are located in the following directories:

/opt/scilibs/FFTW/FFTW3-3.1.2/lib
/opt/scilibs/FFTW/FFTW_mpibull2-<version>/lib
/opt/scilibs/FFTW/FFTW_mpich_ethernet-<version>/lib

Tests are also available in the following directory:

/usr/lib/<version>/test

For more information see www.fftw.org/.

3.12 PETSc

PETSc stands for Portable, Extensible Toolkit for Scientific Computation. PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication (see http://www.mcs.anl.gov/mpi for more details).

The Pets library is available under the following directories for both MPIs:

/opt/scilibs/PETSC/PETSc-2.3.3-p0/mpich_ethernet/lib/linux-intel-opt/
/opt/scilibs/PETSC/PETSc-2.3.3-p0/mpibull2/lib/linux-intel-opt/

For more information see http://www-unix.mcs.anl.gov/petsc/petsc-2/.

3.13 NETCDF

NetCDF (network Common Data Form) allows the management of data input/output. **NetCDF** is an interface for array-oriented data access and is a library that provides an implementation of the interface. The **netCDF** library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data.

The library is located in the following directories:

/opt/scilibs/NETCDF/netCDF_<version>/bin
/opt/scilibs/NETCDF/netCDF_<version>/lib
/opt/scilibs/NETCDF/netCDF_<version>/man

For more information see: http://www.unidata.ucar.edu/software/netcdf/
http://trac.mcs.anl.gov/projects/parallel-netcdf

3.14 METIS and PARMETIS

METIS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in **METIS** are based on the multilevel recursive-bisection, multilevel *k*-way, and multi-constraint partitioning schemes developed in our lab.

ParMETIS is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. **ParMETIS** extends the functionality provided by METIS and includes routines that are especially suited for parallel Adaptive Mesh Refinement computations and large scale numerical simulations.

The libraries for ParmMETIS are located in the following directory:

/opt/scilibs/PARAMETIS/Parmetis_<version>/lib

For more information see http://www-users.cs.umn.edu/~karypis/metis/.

3.15 SciPort

SCIPORT is a portable implementation of **CRAY SCILIB** that provides both single and double precision object libraries. **SCIPORTS** provides single precision and **SCIPORTD** provides double precision.

The libraries for SCIPORT can be found under the following directory:

/opt/scilibs/sciport/sciport-1.0/lib/

For more information see http://www.netlib.org/scilib/sciport.

Chapter 4. Compilers

This chapter describes the following topics:

4.1 Overview

Compilers play an essential role in exploiting the full potential of Itanium® 2 processors. These processors use **EPIC** (Explicit Parallel Instruction set Computing) which enables instructions to be executed in parallel. The parallelism has to be detected and exploited at compiler level. Bull therefore recommends the use of Intel® C/C++ and Intel® Fortran compilers.

GNU compilers are also available. However, these compilers are unable to exploit the **EPIC** instruction set and also any program which uses **MPI_Bull** cannot be compiled\linked with **GNU** products. For **MPI_Bull** programs it is essential that Intel compilers are used.

4.2 Intel Fortran Compiler

The current version of the Intel® Fortran 95 compiler is version 9.

The main features of this compiler are:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Data preloading
- Conditional instruction prediction
- Speculative loading
- Optimization of the software pipeline.

This compiler complies with the Fortran 95 ISO standard. It is also compatible with GNU products. **Emacs** and **gbd** tools can also be used with this compiler. It also supports big endian encoded files. Finally, this compiler allows the execution of applications which combine programs written in C and Fortran.

The compiler supports multithreading functionality:

- OpenMP 2.0 for Fortran is supported. The compiler accepts OpenMP pragmas and generates a multithreaded application.

- Automatic parallelization: a compiler option detects parallelism (in particular in the computation loops) and generates a multithreaded application.

To use the compilers you have to update your environment as described below.

Different versions of the compiler may be installed to ensure compatibility with the compiler version used to compile the libraries and applications on your system.

Note:
It may be necessary to contact the System Administrator to ascertain the location of the compilers on your system. The paths shown in the examples below may vary.

To specify a particular environment use the command below.

```
source /opt/intel/fc/<package_id>/bin/ifortvars.sh
```

For example:

- To use version 9.0.031 of the Fortran compiler:

```
source /opt/intel/fc/9.0.031/bin/ifortvars.sh
```

- To display the version of the active compiler, enter:

```
 ifort --version
```

- To obtain the documentation of the compiler:

```
/opt/intel/fc/9.0.031/doc
```

Remember that if you are using **MPI_Bull** then a compiler version has to be used which is compatible with the compiler originally used to compile the MPI library.

4.3 Intel C/C++ Compiler

The current version of the Intel C/C++ compiler is version 9.

The main features of this compiler are:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Data preloading
- Conditional instruction prediction
- Speculative loading
- Optimization of the software pipeline.

This compiler complies with ISO standard Ansi C/C++ and ISO standard C/C++. It is also compatible with GNU products. A GNU C object or source code can therefore be compiled with an Intel C/C++ compilers. **Emacs** and **gbd** tools can also be used with this compiler.

The compiler supports multithreading functionality:

- OpenMP 2.0 for C/C++ is supported. The compiler accepts OpenMP pragmas and generates a multithreaded application.
- Automatic parallelization: a compiler option detects parallelism (in particular in the computation loops) and generates a multithreaded application.

For more details, visit the Intel web site www.intel.com.

Different versions of the compiler may be installed to ensure compatibility with the compiler version used to compile the libraries and applications on your system.

Note:
It may be necessary to contact the System Administrator to ascertain the location of the compilers on your system. The paths shown in the examples below may vary.

To specify a particular environment use the command below:

```
source /opt/intel/cc/<package_id>/bin/iccvars.sh
```

For example:

- To use version 9.0.037 of the C/C++ compiler:

```
source /opt/intel/cc/9.0.037/bin/iccvars.sh
```

- To display the version of the active compiler, enter:

```
 icc --version
```

- To obtain the documentation of the compiler:

```
/opt/intel/cc/9.0.037/doc
```

Remember that if you are using **MPI_Bull** then a compiler version has to be used which is compatible with the compiler originally used to compile the MPI library.

4.4 Intel Compiler Licenses

Three types of Intel ® compiler licenses are available:

- **Single User:** allows one user to operate the product on multiple computers as long as only one copy is in use at any given time.
- **Node-Locked:** locked to a node, allows any user who has access to this node to operate the product concurrently with other users, limited to the number of licenses purchased.
- **Floating:** locked to a network, allows any user who has access to the network server to operate the product concurrently with other users, limited to the number of licenses purchased.

The node-locked and floating licenses are managed by **FlexLM** from **Macrovision.**

License installation, and **FlexLM** configuration, may differ according to your compiler, the license type, the number of licenses purchased, and the period of support for your product. Please check the Bull Product Designation document delivered with your compiler and follow the instructions contained therein.

4.5 Intel Math Kernel Library Licenses

Intel Math Kernel Library licenses are required for each compile node on which you compile with **MKL**. However, the runtime libraries which are used on the compute nodes do not require a license fee.

4.6 GNU Compilers

GCC, a collection of free compilers that can compile both C/C++ and Fortran, is part of the installed Linux distribution.

Chapter 5. The User's Environment

This chapter describes how to access the HPC environment, how to use file systems, and how to use the modules package to switch and compare environments:

5.1 Cluster Access and Security

Typically, users connect to and use a HPC cluster as described below:

- Users log on to the HPC platform either through Service Nodes or through the Login Node when the configuration includes these special Login Node(s). Once logged on to a node, users can then launch their jobs.
- Compilation is possible on all nodes which have compilers installed on them. The best approach is that compilers reside on Login Nodes, so that they do not interfere with performance on the Compute Nodes.

5.1.1 Using ssh (Secure Shell)

The **ssh** command is used to access a cluster node.

Syntax:

```
ssh [-l login_name] hostname | user@hostname [command]

ssh [-afgknqstvxACNTX1246] [-b bind_address] [-c cipher_spec]
    [-e escape_char] [-i identity_file] [-l login_name] [-m mac_spec]
    [-o option] [-p port] [-F configfile] [-L port:host:hostport]
    [-R port:host:hostport] [-D port] hostname | user@hostname [command]
```

ssh (ssh client) can also be used as a command to log onto a remote machine and to execute commands on it. It replaces **rlogin** and **rsh**, and provides secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel. **ssh** connects and logs onto the specified hostname. The user must verify his/her identity, using the appropriate protocol, before being granted access to the remote machine.

5.2 Global File Systems: NFS / Lustre

Two major kinds of file systems are generally used in a HPC environment:
NFS (distributed file system) and **LUSTRE** (parallel file system).

Lustre is an Open Source product under a GPL License. **Lustre** is specially designed for the needs of high performance systems with a large data bandwidth. This design means that Lustre is able to take full advantage of **QSNET^{II}** high flow, weak latency interconnect networks so that metadata and data is transferred efficiently.

Using Lustre

Data and metadata is stored under **ldiskfs** local files. **ldiskfs** is an **ext3** file system with special patches for **Lustre**.

QSNET^{II} networks allow a flow of 900 MB/s for one rail (link). This flow may be restricted by the flow of the disk bay and depends upon the Input/Output typology.

Lustre is usually used as follows:

- Each user has a private directory under **/home_nfs.**
- Each user has a private directory under the **Lustre** file system. Generally data under a **Lustre** file system is not saved, and can be deleted by the cluster's administrator whenever he needs to. If you want to save your data, you have to copy it using **NFS**.
- The **Lustre** File System is mounted following the user's request on the specified computation nodes.
- Two possible ways of running an application on a HPC system are:
  - The code is within the **Lustre** system (it must have been copied from NFS before launch) and the results are generated under **Lustre**.
  - The code is within NFS and the results are generated within **Lustre** (output files must be defined for the application in **/mnt/lustre/user's directory**).

For example:

To copy a **NFS** file into a **Lustre** file system using **prun**, enter:

```
prun -p my_partition -N1 -n1 cp -r
~/home_nfs/'whoami'/pathname /mnt/lustre/'whoami'/pathname
```

For details about Lustre's administration and operation refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

For information about optimizing the file system refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER).

5.3 Environment Modules

Environment modules provide a great way to customize your shell environment easily, particularly on the fly.

For instance an environment can consist of one set of compatible products including a defined release of a FORTRAN compiler, a C compiler, a debugger and mathematical libraries. In this way you can easily reproduce trial conditions, or use only proven environments.

The **Modules** environment is a program that can read and list module files returning commands, suitable for the shell to interpret, and most importantly for the **eval** command. **Modulefiles** is a kind of flat database which uses files.

In UNIX a child process cannot modify its parent environment.
So how does Modules do this? Modules parses the given modules file and produces the appropriate shell commands to **set/unset/append/un-append** onto an environment variable. These commands are eval'd by the shell. Each shell provides some mechanism where commands can be executed and the resulting output can, in turn, be executed as shell commands. In the C-shell & Bourne shell and derivatives this is the **eval** command.

This is the only way that a child process can modify the parent's (login shell) environment. Hence the module command itself is a shell alias or function that performs these operations. To the user, it looks just like any other command.

The **module** command is only used in the development environment and not in other environments such as that for administration node.

More details are available at http://modules.sourceforge.net/

5.3.1 Using Modules

The following command gives the list of available modules on this cluster.

```
module avail

------------------------ /opt/modules/version ------------------------
3.1.6

------------------- /opt/modules/3.1.6/modulefiles -------------------
dot          module-info null
module-cvs   modules     use.own

---------------------- /opt/modules/modulefiles ----------------------
oscar-modules/1.0.3 (default)
```

Modules available for the user are listed under the line **/opt/modules/modulefiles.**

To load a module the command is:

```
module load module_name
```

To verify the loaded modules list the command is:

```
module list
```

Using the `avail` command it is possible that some modules will be marked *(default):*

```
module avail
```

These modules are those which have been loaded without the user specifying a module version number. For example the following commands are the same:

```
module load configuration

module load configuration/2
```

Three configurations have been created. These configurations are modules which load other modules automatically.

For example the number 2 configuration includes:

- Intel Fortran compiler version 8.0.049
- Intel C compiler version 8.0.071
- Intel debugger version 8.1.3
- MKL version 7.0.017

Configuration/1	intel_fc –version 8.0.046 intel_cc –version 8.0.066 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/2	intel_fc –version 8.0.049 intel_cc –version 8.0.071 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/3	intel_fc –version 8.0.061 intel_cc –version 8.0.071 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/4	intel_fc –version 8.0.019 intel_cc –version 8.0.022 intel_db –version 8.1.3 intel_mkl –version 7.0.017

Table 5-1. Examples of different module configurations

The use of the **load** command in the module configuration context changes the user "prompt" adding the configuration name.

The **module unload** command unloads one module.

The **module purge** command clears all the modules from the environment.

```
module purge
```

By design two "configuration" modules can not be loaded simultaneously. The loading of a "configuration" module unloads the previous one.

It is not possible to load the two modules **intel_cc** or **intel_fc** at the same time because it causes conflicts.

5.3.2 Setting Up the Shell RC Files

Here's a quick tutorial on Shell rc (run-command) files. When a user logs in and if they **have /bin/csh(/bin/sh)** as their shell, the first **rc** fire to be parsed by the shell is **/etc/csh.login** & **/etc/csh.cshrc (/etc/profile)** (the order is implementation dependent), and then the user's $HOME/.cshrc ($HOME/.kshenv) and finally $HOME/.login ($HOME/.profile).

All the other login shells are based on **/bin/csh** and **/bin/sh** with additional features and **rc** files. Certain environment variables and aliases (functions) need to be set for Modules to work correctly. This is handled by the Module init files in **/opt/modules /default/init**, which contains separate init files for each of the various supported shells, where the default is a symbolic link to a module command version.

Skeleton Shell RC ("Dot'') Files

The skeleton files provide a "default'' environment for new users when they are added to your system, this can be used if you do not have the time to set them up individually. The files are usually placed in **/etc/skel** (or wherever you specified with the --with-skel-path=<path> option to the configuration script), and contains a minimal set of "dot'' files and directories that every new user should start with.

The skeleton files are copied to the new user's $HOME directory with the "**-m**'' option added to the "**useradd**'' command. A set of sample "dot'' files are located in **./etc/skel**. Copy everything but the .*.in and CVS files and directories to the skeleton directory. Edit and tailor for your system.

If you have a pre-existing set of skeleton files, then make sure the following minimum set exists: **.cshrc**, **.login**, **.kshenv**, **.profile**. These can be automatically updated with the command:

```
env HOME=/etc/skel/opt/modules/default/bin/add.modules
```

Inspect the new "dot'' files and if they are OK, then remove all the .*.old (original) files. An alternative way of setting-up the users' dot files can be found in ./ext.
This model can be used with the --with-dot-ext configure option.

User Shell RC ("Dot'') Files

The final step for a functioning modules environment is to modify the user "dot'' files to source the right files. One way to do this is to put a message in the **/etc/motd** telling each user to run the command:

```
/opt/modules/default/bin/add.modules
```

This is a script that parses their existing "dot'' files prepending the appropriate commands to initialize the Modules environment.

The user can re-run this script and it will find and remember what modules they initially loaded and then strip out the previous module initialization and restore it with an upgraded one.

If the user lacks a necessary "dot'' file, the script will copy one over from the skeleton directory. The user will have to logout and login for it to come into effect.
Another way is for the system administrator to "su - username" to each user and run it interactively. The process can be semi-automated with a single line command that obviates the need for direct interaction:

```
su - username -c "yes | /usr/local/Modules/default/bin/add.modules"
```

Power users can create a script to directly parse the **/etc/passwd** file to perform this command. Otherwise, just copy the passwd file and edit it to execute this command for each valid user.

5.4 Module Files

Once the above steps have been performed, then it is important to have module files in each of the modulefiles directories. For example, the following module files will be installed:

```
--------- /opt/modules/3.0.9-rko/modulefiles ----------
dot          module-info modules     null        use.own
```

If you do not have your own module files in **/opt/modules/modulefiles** then copy "null'' to that directory. On some systems an empty modulefiles directory will cause a core dump, whilst on other systems there will be no problem. Use **/opt/modules/default/modulefiles/modules** as a template for creating your own module files.

For more information run:

```
module load modules
```

You will then have ready access to the module(1) modulefile(4) man pages, as well as the versions directory. Study the man pages carefully.
The version directory may look something like this:

```
---------------- /opt/modules/versions ----------------
3.0.5-rko 3.0.6-rko 3.0.7-rko 3.0.8-rko 3.0.9-rko
```

The model you should use for modulefiles is "name/version''. For example, **/opt/modules/modulefiles** directory may have a directory named "firefox'' which contains the following module files: 301, 405c, 451c, etc.
When it's displayed with "module avail'' it looks something like this:

```
firefox/301
firefox/405c
firefox/451c(default)
firefox/45c
firefox/46
```

The default is established with **.version** file in the firefox directory and it looks something like this:

```
#%Module1.0############################################################
##
## version file for Firefox
##
set ModulesVersion      "451c"
```

If the user does "module load firefox'', then the default firefox/451c will be used. The default can be instantly changed by editing the **.version** file to point to a different module file in that directory. If no **.version** file exists then Modules will just use the last module in the alphabetical ordered directory listing as the default.

5.4.1 Upgrading via the Modules Command

The theory is that Modules should use a similar package/version locality as the package environments it helps to define. Switching between versions of the module command should be as easy as switching between different packages via the module command. Suppose there is a change from 3.0.5-rko to version 3.0.6-rko. The goal is to semi-automate the changes to the user "dot'' files so that the user is oblivious to the change.

The first step is to install the new module command & files to **/opt/modules/3.0.6-rko/**. Test it out by loading with "module load modules 3.0.6-rko". You may get an error like: 3.0.6-rko (25):ERROR:152: Module 'modules' is currently not loaded. This is OK and should not appear with future versions.

Make sure you have the new version with "module --version". If it seems stable enough, then advertise it to your more adventurous users. Once you are satisfied that it appears to work adequately well, then go into **/opt/moduless** remove the old "default" symbolic link to the new versions.

For example:

```
cd /opt/modules
rm default; ln -s 3.0.6-rko default
```

This new version is now the default and will be referenced by all the users that log in and by those that have not loaded a specific module command version.

5.5 The Module Command

Synopsis

```
module [ switches ] [ sub-command ] [ sub-command-args ]
```

The **Module** command provides a user interface to the Modules package. The Modules package provides for the dynamic modification of the user's environment via *modulefiles*.

Each *modulefile* contains the information needed to configure the shell for an application. Once the Modules package is initialized, the environment can be modified on a per-module basis using the module command which interprets *modulefiles*. Typically *modulefiles* instruct the **module** command to alter or to set shell environment variables such as PATH, MANPATH, etc. *modulefiles* may be shared by many users on a system and users may have their own collection to supplement or replace the shared *modulefiles*.

The *modulefiles* are added to and removed from the current environment by the user. The environment changes contained in a *modulefile* can be summarized through the module command as well. If no arguments are given, a summary of the module usage and sub-commands are shown.

The action for the module command to take is described by the sub-command and its associated arguments.

5.5.1 modulefiles

modulefiles are the files containing **TCL** code for the Modules package.

modulefiles are written in the Tool Command Language, **TCL**(3) and are interpreted by the modulecmd program via the module(1) user interface. **modulefiles** can be loaded, unloaded, or switched on-the-fly while the user is working.

A modulefile begins with the magic cookie, '#%Module'. A version number may be placed after this string. The version number is useful as the format of **modulefiles** may change. If a version number doesn't exist, then modulecmd will assume the modulefile is compatible with the latest version. The current version for **modulefiles** will be 1.0. Files without the magic cookie will not be interpreted by modulecmd.

Each modulefile contains the changes to a user's environment needed to access an application. **TCL** is a simple programming language which permits **modulefiles** to be arbitrarily complex, depending on the needs of the application and the modulefile writer. If support for extended tcl (tclX) has been configured for your installation of modules, you may use all the extended commands provided by tclX, too. **modulefiles** can be used to implement site policies regarding the access and use of applications.

A typical **modulefiles** file is a simple bit of code that sets or adds entries to the PATH, MANPATH, or other environment variables. **TCL** has conditional statements that are evaluated when the modulefile is loaded. This is very effective for managing path or environment changes due to different OS releases or architectures. The user environment information is encapsulated into a single modulefile kept in a central location. The same modulefile is used by all users independent of the machine. So, from the user's perspective, starting an application is exactly the same regardless of the machine or platform they are on.

modulefiles also hide the notion of different types of shells. From the user's perspective, changing the environment for one shell looks exactly the same as changing the environment for another shell. This is useful for new or novice users and eliminates the need for statements such as "if you're using the C Shell do this ..., otherwise if you're using the Bourne shell do this ..." Announcing and accessing new software is uniform and independent of the user's shell. From the modulefile writer's perspective, this means one set of information will take care of all types of shells.

5.5.2 Modules Package Initialization

The Modules package and the module command are initialized when a shell-specific initialization script is sourced into the shell. The script creates the module command as either an alias or function, creates Modules environment variables, and saves a snapshot of the environment in ${HOME }/.modulesbeginenv. The module alias or function executes the modulecmd program located in ${MODULESHOME }/bin and has the shell evaluate the command's output. The first argument to modulecmd specifies the type of shell.

The initialization scripts are kept in ${MODULESHOME }/init/shellname where shellname is the name of the sourcing shell. For example, a C Shell user sources the ${MODULESHOME }/init/csh script. The **sh**, **csh**, **tcsh**, **bash**, **ksh**, and **zsh** shells are all supported by **modulecmd**. In addition, python and perl "shells" are supported which writes the environment changes to stdout as python or perl code.

5.5.3 Examples of Initialization

In the following examples, replace ${MODULESHOME } with the actual directory name.

C Shell initialization (and derivatives)

```
source ${MODULESHOME }/init/csh module load modulefile modulefile
```

Bourne Shell (sh) (and derivatives)

```
${MODULESHOME }/init/sh module load modulefile modulefile
```

Perl

```
require "${MODULESHOME }/init/perl"; &module("load modulefile modulefile ");
```

5.5.4 Modulecmd Startup

Upon invocation modulecmd sources **rc** files which contain global, user and *modulefile* specific setups. These files are interpreted as modulefiles.

Upon invocation of modulecmd module RC files are sourced in the following order:

1. Global RC file as specified by ${MODULERCFILE } or
   ${MODULESHOME }/etc/rc
2. User specific module RC file ${HOME }/.modulerc
3. All .module rc and .version files found during modulefile searches.

5.5.5 Module Command Line Switches

The module command accepts command line switches as its first parameter. These may be used to control output format of all information displayed and the module behavior in the case of locating and interpreting module files.

All switches may be entered either in short or long notation. The following switches are accepted:

--force, -f

Force active dependency resolution. This will result in modules found on a prereq command inside a module file being loaded automatically. Unloading module files using this switch will result in all required modules which have been loaded automatically using the -f switch being unloaded. This switch is experimental at the moment.

--terse, -t

Display avail and list output in short format.

--long, -l

Display avail and list output in long format.

--human, -h

Display short output of the avail and list commands in human readable format.

--verbose, -v

Enable verbose messages during module command execution.

--silent, -s

Disable verbose messages. Redirect **stderr** to **/dev/null** if **stderr** is found not to be a **tty**. This is a useful option for module commands being written into **.cshrc** , **.login** or .profile files, because some remote shells (e.g. **rsh** (1)) and remote execution commands (e.g. **rdist**) get confused if there is output on **stderr**.

--create, -c

Create caches for module **avail** and module **apropos** . You must be granted write access to the ${MODULEHOME }/*modulefiles/* directory if you try to invoke module with the -c option.

--icase, -i

This is a case insensitive module parameter evaluation. Currently only implemented for the module apropos command.

--userlvl <lvl>, -u <lvl>

Set the user level to the specified value. The argument of this option may be one of:

novice,	nov Novice
expert,	exp Experienced module user
advanced,	adv Advanced module user

5.5.6 Module Sub-Commands

Print the use of each sub-command. If an argument is given, print the Module specific help information for the *modulefile*.

```
help [modulefile...]
```

Load **modulefile** into the shell environment.

```
load modulefile [modulefile...]
add modulefile [modulefile...]
```

Remove *modulefile* from the shell environment.

```
unload modulefile [modulefile...]
rm modulefile [modulefile...]
```

Switch loaded *modulefile*1 with *modulefile*2.

```
switch modulefile1 modulefile2
swap modulefile1 modulefile2
```

Display information about a *modulefile*. The display sub-command will list the full path of the *modulefile* and all (or most) of the environment changes the *modulefile* will make when loaded. (It will not display any environment changes found within conditional statements).

```
display modulefile [modulefile...]
```

List loaded modules.

```
show modulefile [modulefile...]
list
avail [path...]
```

List all available *modulefiles* in the current MODULEPATH. All directories in the MODULEPATH are recursively searched for files containing the *modulefile* magic cookie. If an argument is given, then each directory in the MODULEPATH is searched for *modulefiles* whose pathname match the argument. Multiple versions of an application can be supported by creating a subdirectory for the application containing *modulefiles* for each version.

```
use directory [directory...]
```

Prepend directory to the MODULEPATH environment variable. The --append flag will append the directory to MODULEPATH.

```
use [-a|--append] directory [directory...]
```

Remove directory from the MODULEPATH environment variable.

```
unuse directory [directory...]
```

Attempt to reload all loaded *modulefiles*. The environment will be reconfigured to match the saved ${HOME }/**.modulesbeginenv** and the *modulefiles* will be reloaded. The `update` command will only change the environment variables that the *modulefiles* set.

```
update
```

Force the Modules Package to believe that no modules are currently loaded.

```
clear
```

Unload all loaded *modulefiles*.

```
purge
```

Display the *modulefile* information set up by the module-`whatis` commands inside the specified *modulefiles*. If no *modulefiles* are specified, all the whatis information lines will be shown.

```
whatis [modulefile [modulefile...]]
```

Searches through the `whatis` information of all *modulefiles* for the specified string. All module `whatis` information matching the search string will be displayed.

```
apropos string
keyword string
```

Add *modulefile* to the shell's initialization file in the user's home directory. The startup files checked are .cshrc, .login, and .csh_variables for the C Shell; .profile for the Bourne and Korn Shells; .bashrc, .bash_env, and .bash_profile for the GNU Bourne Again Shell; .zshrc, .zshenv, and .zlogin for zsh. The .modules file is checked for all shells. If a 'module load' line is found in any of these files, the *modulefile*(s) is(are) appended to any existing list of *modulefiles*. The 'module load' line must be located in at least one of the files listed above for any of the 'init' sub-commands to work properly. If the 'module load' line is found in multiple shell initialization files, all of the lines are changed.

```
initadd modulefile [modulefile...]
```

Does the same as initadd but prepends the given modules to the beginning of the list. initrm *modulefile* [*modulefile*...] Remove *modulefile* from the shell's initialization files.

```
initprepend modulefile [modulefile...]
```

Switch *modulefile*1 with *modulefile*2 in the shell's initialization files.

```
initswitch modulefile1 modulefile2
```

List all of the *modulefiles* loaded from the shell's initialization file.

```
initlist
```

Clear all of the *modulefiles* from the shell's initialization files.

```
initclear
```

5.5.7 Modules Environment Variables

Environment variables are unset when unloading a *modulefile*. Thus, it is possible to load a *modulefile* and then unload it without having the environment variables return to their prior state.

MODULESHOME:

This is the location of the master Modules package file directory containing module command initialization scripts, the executable program modulecmd, and a directory containing a collection of master *modulefiles*.

MODULEPATH:

This is the path that the module command searches when looking for *modulefiles*. Typically, it is set to the master *modulefiles* directory, ${MODULESHOME }/*modulefiles*, by the initialization script. MODULEPATH can be set using 'module use' or by the module initialization script to search group or personal *modulefile* directories before or after the master *modulefile* directory.

LOADEDMODULES

A colon separated list of all loaded *modulefiles*.

LOADED_*MODULEFILES*

A colon separated list of the full pathname for all loaded *modulefiles*.

MODULESBEGINENV

The filename of the file containing the initialization environment snapshot.

Files

/opt

The MODULESHOME directory.

${MODULESHOME}/etc/rc

The system-wide modules rc file. The location of this file can be changed using the MODULERCFILE environment variable as described above.

${HOME}/.modulerc

The user specific modules rc file.

${MODULESHOME}/*modulefiles*

The directory for system-wide *modulefiles*. The location of the directory can be changed using the MODULEPATH environment variable as described above.

${MODULESHOME}/bin/modulecmd

The *modulefile* interpreter that gets executed upon each invocation of a module.

${MODULESHOME}/init/shellname

The Modules package initialization file sourced into the user's environment.

${MODULESHOME}/init/.modulespath

The initial search path setup for module files. This file is read by all shell init files.

${MODULEPATH}/.moduleavailcache

File containing the cached list of all *modulefiles* for each directory in the MODULEPATH (only when the avail cache is enabled).

${MODULEPATH}/.moduleavailcachedir

File containing the names and modification times for all sub-directories with an avail cache.

${HOME}/.modulesbeginenv

A snapshot of the user's environment taken when Modules are initialized. This information is used by the module update sub-command.

Chapter 6. Launching an Application

This chapter describes the following topics:

6.1 Launching the Application without a Batch Manager

There are different ways of launching the application on Bull HPC platforms, without using a batch manager. These vary according to platform and application type. Refer to the table on the next page for information on the different possibilities that are available.

A second step is to ensure that once launched the execution is fully optimized. The tools and commands to be used to do this are also indicated. It is possible that the system administrator may have to intervene in order to allocate the resources for the application.

For more information on where to find these tools and how to use them, refer to the rest of this chapter and the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER). For more information on the commands for the **pdsh** shell utility, refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

Note:

For more information on **mprun**, used in a single node parallel environment, and **mpibull2-launch**, a meta-launcher which helps users retain their launching commands when changing MPI environments and process managers, refer to chapter 2 of this manual.

Platform	Application		Launching tool
Clusters with no Resource Manager	Serial		none
	Parallel	OpenMP	none
		MPI Bull1	mprun
		MPIBull2	mpiexec/mpirun (MPD)
Clusters with Quadrics RMS	Serial		rmsexec
	Parallel	OpenMP on one node	allocate prun –c <no. of CPUs>
		MPI	prun
		Hybrid (MPI + OpenMP)	prun –c <no. of CPUs per MPI task>
Clusters with SLURM	Serial		srun
	Parallel	OpenMP on one node	srun –A srun –c <no. of CPUs >
		MPI	srun
		Hybrid (MPI + OpenMP)	srun –c <no. of CPUs per MPI task>

Table 6-1. Launching tools for different clusters

Note:
There are memory access differences for the different hardware architectures covered by this manual. **NovaScale 5xxx/6xx0 Series** platforms use the Quad Brick Board (**QBB**) hardware architecture with Non Uniform Memory Access (**NUMA**). Symmetric Multiprocessing (**SMP**) is used for **NovaScale 4xx0 Series. NovaScale 3005 Series** platforms have a very low **NUMA** factor which is disabled by default.

In **SMP** platforms the memory access time is stable for all processors, and the Quad Brick Board hardware model is not used. The term **QBB** for these platforms refers to the set of sockets which are attached to the Scalable Node Controller (**SNC**) on the system board for **NovaScale 4xx0** platforms, and to the Node Controller (**NC**) on the system board for **NovaScale 3005** platforms. This means that 1 QBB, which may include 1-4 single processors, is possible for the **NovaScale 4xx0** platforms, whilst for the **NovaScale 3005 Series** 2 QBBs are possible, each of which may house 1-2 dual core sockets.

6.1.1 NUMACTL

Numactl is dedicated to **single-NUMA** systems. The granularity is restricted to the QBB level for each node. The following example shows a node with 16 CPUs.

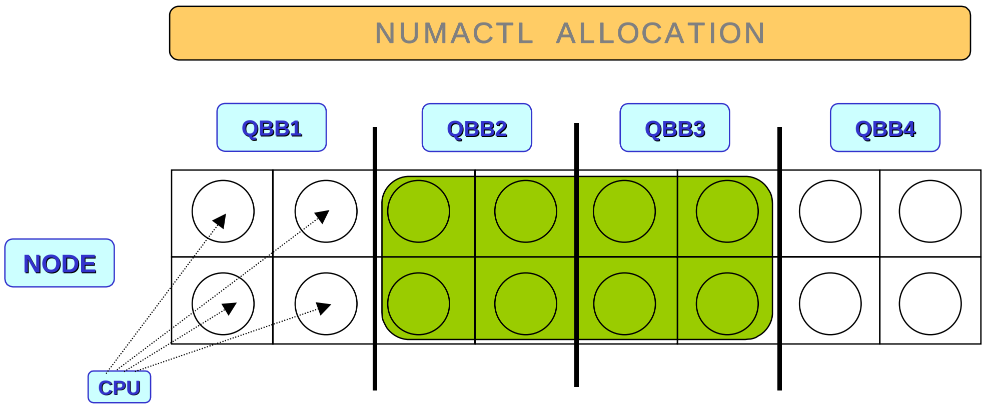


Figure 6-1. Numactl QBB application

Numactl is able to define an execution area for an application, in this example QBB2 and QBB3 (2 * 4 CPUs) are allocated.

6.1.1.1 Using Libnuma and Numactl

Important:

The scope of the **numactl** command is a mono numa configuration, with 1 to 8 QBBs that is to say, one node for a HPC cluster.

In the following paragraph concerning the numactl command, "node" means a QBB in the numa configuration.

Libnuma is a library that offers a simple programming interface to the Symmetric Multi Processing NUMA policy supported by the Linux kernel. In a **NUMA** architecture, memory areas have different latencies or bandwidths according to which CPUs they are accessed from.

Available policies are page interleaving, preferred node allocation, local allocation, allocation only on specific nodes. The binding of threads to specific nodes is also possible. All policies exist per thread and are inherited by children.

For setting global policy per process it is easiest to run **Libnuma** using the **numactl** utility. This library can be used for a more fine grained policy inside an application. Outside the application the policy applies to all the memory of the process, whereas inside you can use it for each memory zone.

The granularity level of allocation for **numactl** is the node i.e. a QBB.

All numa memory allocation policies only take effect when pages are actually faulted into the address space of a process by accessing them. The **numa_alloc_*** functions take care of this automatically.

Before any other calls in this library can be used **numa_available** must be called. When it returns a negative value all other functions in this library are undefined.

numactl runs processes with a specific **NUMA** scheduling or memory placement policy. The policy is set for a command and inherited by all of its children. In addition **numactl** can set a persistent policy for shared memory segments or files.

The most common policy settings are:

- -interleave=nodes, -i nodes

Sets an memory interleave policy. Memory will be allocated using a round robin algorithm on nodes. When memory cannot be allocated on the current interleave, the target falls back to other nodes.

- -membind=nodes, -m nodes

Only allocates memory from the specified nodes. Allocation will fail when there is not enough memory available on these nodes.

- -cpubind=nodes, -c nodes

Only executes process on the CPUs of the nodes specified.

Note:

It is possible that this command may conflict with the usage of **CPUSETS**. As an alternative it is suggested that you use **numactl** to set your **mempolicy** set rather than **CPUSETS** and/or **taskset** or **sched_affinity** for CPU bindings.

- -localalloc, -l

Always allocates locally on the current node.

Example

To run a program which allocates memory using a round robin allocation on 4 nodes of a 16 CPU NovaScale server, enter:

```
numactl -i0,1,2,3 program_name
```

For more information refer to the **numa** man pages.

Libnuma comes under the GNU Lesser General Public License, v2.1.

6.1.2 The PTOOLS and CPUSET Package

The **Ptools package** includes the **pexec** and the **pcreate** commands which can be used to create and to execute cpusets, and also to allocate resources inside an HPC node. The minimum granularity level is the CPU within a QBB. In the following example we have:

- CPUSET 1 with 2 CPUs on QBB1, 2 CPUs on QBB2 and 2 CPUs on QBB3,

- CPUSET 2 with 2 CPUs on QBB2 and 2 CPUs on QBB3,
- CPUSET 3 with 4 CPUs on QBB4.

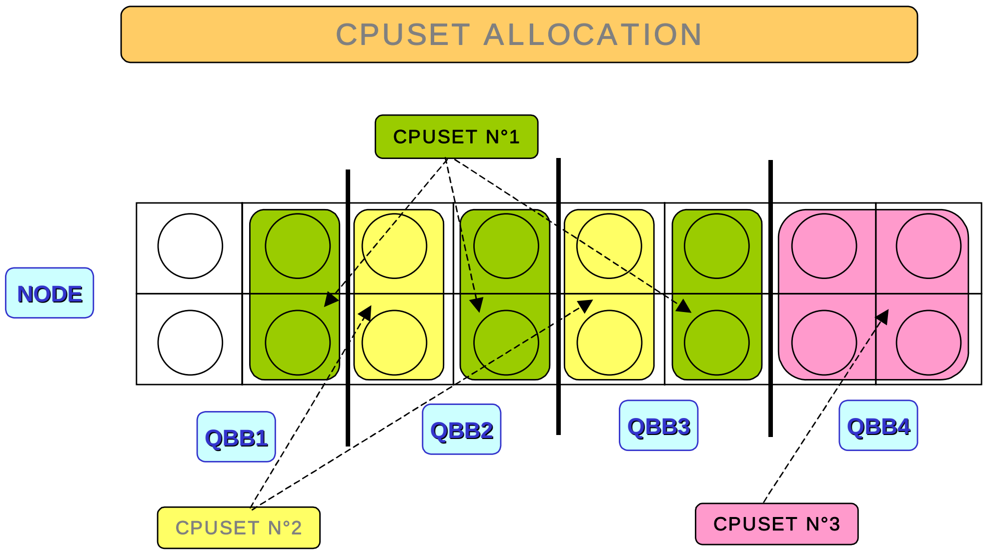


Figure 6-2. CPUSET allocation

6.1.2.1 Using Ptools and CPUSET

CPUSET is a feature of the Bull Linux kernel, which lets you define execution areas inside a multiprocessor system. The execution of each program will be limited to these predefined areas. These execution areas are called **cpusets**.

Cpusets can form a nested hierarchy meaning that **cpusets** can be created inside a **cpuset**.

Cpusets are used:

- To offer some kind of partitioning for multiprocessor systems.
- To ensure the highest performance for the execution of an application, especially on systems with a complex topology such as NUMA systems.

Cpusets also changes the way you map processes on specific processors. When a task uses the **sched_setaffinity** system call, the list of processors specified for this system call is interpreted to be used *inside* the **cpuset** in which the application is running. For example, if an application running inside a **cpuset** with processors 4, 5, 6 and 7 wants to bind one of its processes to the processor 0, the process will actually be bound to processor 4. This feature allows you to run several applications at the same time, and to finely control which processors their tasks are running on.

Bull provides the **ptools** suite to create and run **cpusets**.

ptools consists of the following:

pcreate	To create cpusets.
pexec	To create a cpuset and run an application inside it. The cpuset is destroyed when the application is completed.
passign	To move a task inside a cpuset.

pdestroy	To destroy a cpuset.
pls	To list existing cpusets.
pplace	To finely tune the binding of threads and processes for an application. See the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER) for more details.

When a **cpuset** is created, a list of processors must be chosen. Several flags can also be set for each **cpuset**:

strict	Also called **cpu_exclusive**. This **cpuset** will not share its processors with other **cpusets** that have the same parent **cpuset**.
autoclean	To automatically remove a **cpuset** from a system and to free its resources when it becomes unused. That is to say when all the applications running inside the cpuset are finished.

Example

```
pexec –np <nb_cpus> --strict <my_app>
```

```
[root@nsadmin root]# pexec -np 2 --strict ./myapp

Created /dev/cpuset/cpuset0

Myapp running..
```

For more information refer to the installed man pages of **pexec**, **pcreate** and **passign**.

6.2 Quadrics Resource Management System

The Quadrics Resource Management System **(RMS)** includes the **prun** command to define partitions and to run jobs in a HPC cluster. One partition can lie across several nodes, as is the case for the "PARALLEL partition N°1" in the following figure:

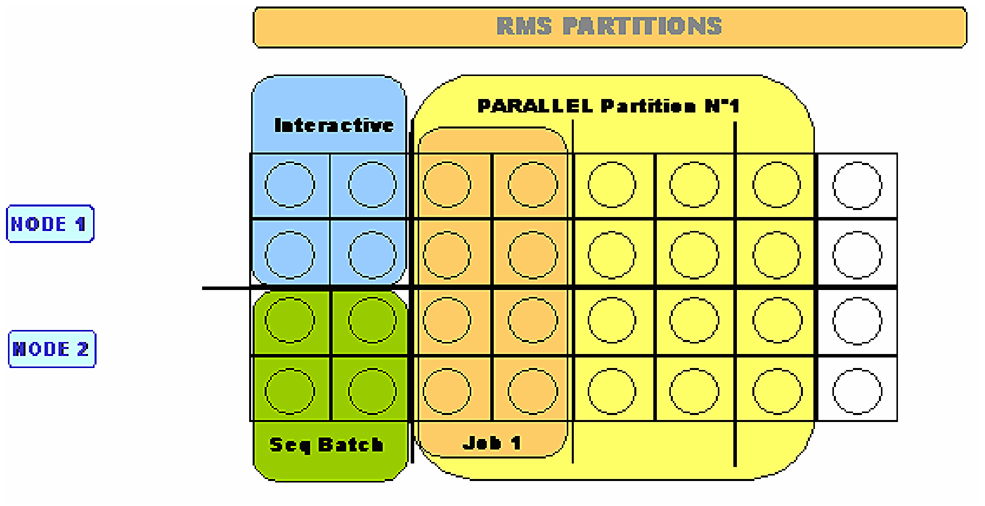


Figure 6-3. RMS Partitions

6.2.1 Using Quadrics RMS

The key to achieving high-levels of performance for a large-scale parallel application is to dedicate specific resources (CPUs, memory, network bandwidth and local I/O capability) to its execution. Quadrics RMS enables a system administrator to efficiently manage these resources to achieve maximum performance. Nodes can be configured into mutually exclusive sets known as partitions; these may each provide a specific system service. For example, your system could have an interactive partition for conventional UNIX processes and program development, a sequential batch partition, and a parallel partition running the RMS gang scheduler. Free cycles on the interactive partition could be used by sequential batch jobs running from a low priority queue. Plus the system may be configured to allow certain users to run high-priority interactive jobs during working hours.

Parallel programs under Quadrics **RMS** are managed by a controlling process **prun** and have application processes distributed over the nodes of a partition. Each process is executed by dedicated CPUs. You choose how many are required for each process, and how they are distributed over multi-CPU nodes.

The administrator of an RMS system controls how the nodes are configured into partitions, how they change, and who can access each partition and the level of resources that they use.

RMS also provides accounting facilities.

The user commands required to launch an application with **RMS** are as follows:

prun
The **prun** program loads and runs parallel programs. It can also run multiple copies of a sequential program.

rmsexec
The **rmsexec** program runs a sequential program on a lightly loaded node.

6.2.2 Prun

The main options for **prun** are as follows:

-n <procs>	Specifies the number of processes required
-p <partitions>	Specifies the partition on which to run the program
-Rrails=<nbrails>	Gives the number of rails to use
-N<nodes>	Specifies the number of nodes required
-B<base>	Specifies the first node to use
-s	Prints statistics as the job exits
-t	Prefix output with the process number
-o<output_file.txt>	Redirects output to **output_file.txt**
-e<err_file.txt>	Redirects errors to **err_file.txt**

6.2.3 Rmsexec

The **rmsexec** program provides a mechanism for running sequential programs on lightly loaded nodes with free memory or low CPU usage. It locates a suitable node and then runs the program on it. The user can select a node from a specific partition (of type login or general) with the **-p** option. Without the **-p** option **rmsexec** uses the default load-balancing partition (specified with the **lbal**-partition attribute in the attributes table). In addition, the hostname of the node can be specified explicitly. The request will fail if this node is not available according to the access rights of the user. System administrators may select any node.

Note:

This load balancing service may not be available on all types of partitions.

The main options for **rmsexec** are the following ones:

```
rmsexec [-hv] [-p partition] [-s stat] [hostname] program [args ...]
```

Use the **-h** option to get a list of the available options and valid arguments.

Selecting a Node

rmsexec restricts its search for a lightly loaded node to the partitions you are entitled to use (as defined by the system administrator). You can restrict the search still further by naming a particular partition with the **-p** option, as shown in the following example:

```
$ rmsexec -p parallel myseqprog
```

You can also request a processor on a specific node. The following example requests the node atlas2:

```
$ rmsexec atlas2 myseqprog
```

6.2.4 rinfo

rinfo is a **RMS** command used on HPC platforms with Quadrics Interconnects and which provides you with a global overview of the partitions defined by **RMS** on a cluster including the number of CPUs and machines within it. **rinfo** will also indicate the number of CPUs used when an application is executed within a partition and the state of affairs for the active applications.
This command can also be used to obtain further information on the topology of the cluster.

Example:

```
$ rinfo
```

```
---------------------------------------------------------------------
MACHINE      CONFIGURATION
nsad         day

PARTITION    CPUS    STATUS       TIME    TIMELIMIT       NODES
root         28                                           ns[13-15]
nsad
part1        0/8     running   1:00:07:02              ns[13-14]
part2        ??/0    down      --:--
---------------------------------------------------------------------
```

In the example above the cluster consists of 28 processors and 3 nodes: ns 13, ns 14 and ns 15. The first RMS partition is shown as 'part1' and consists of 2 nodes (ns13 and ns14) and 8 CPUs and its status is 'running' which means that it can be used.

The second partition is 'part2' and its status is 'down', with no nodes allocated, which means that it cannot be used.

6.2.5 More RMS Information

For more information, see Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) or refer to the **RMS** *User's Guide* and the Quadrics web site at http://www.quadrics.com

See the "*RMS Reference Manual*" at http://www.quadrics.com for details about other RMS commands.

6.3 SLURM Resource Management Utilities

As a cluster resource manager, SLURM has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates conflicting requests for resources by managing a queue of pending work.

Users interact with **SLURM** through various command line utilities:

- **SRUN** for submitting a job for execution and optionally controlling it interactively.
- **SBCAST** to transmit a file to all nodes running a job.
- **SCANCEL** for terminating a pending or running job.
- **SQUEUE** for monitoring job queues.
- **SINFO** for monitoring partition and overall system state.
- **SACCT** displays data for all jobs and job steps in the SLURM accounting log.
- **Global Accounting API** for merging the data from the **LSF** accounting file and the SLURM accounting file into a single record.

Note:

There is a general explanation of each available commanding the following sections. For complete and detailed information please refer to the man pages. For example, man srun

Note:
See the HPC BAS4 *Application Tuning Guide* for information on the Consumable Resource Scheduling Policy using the CPU Consumable Resource node allocation plug-in.

6.3.1 SRUN

SRUN submits jobs to run under **SLURM** management. **SRUN** can:

- Submit a batch job and then terminate
- Submit an interactive job and then persist to shepherd the job as it runs
- Allocate resources to a shell and then spawn that shell for use in running subordinate jobs.

SLURM associates every set of parallel tasks ("*job steps*") with the **SRUN** instance that initiated that set, and SRUN provides comprehensive control over node choice and I/O redirection for the parallel job.

6.3.1.1 SRUN Roles and Modes

SRUN executes tasks ("*jobs*") in parallel on multiple compute nodes at the same time (on machines where SLURM manages the resources). **SRUN** options allow the User to both:

- Specify the parallel environment for job(s), such as the number of nodes used, node partition, distribution of processes among nodes, and total time.
- Control the behavior of a parallel job as it runs, such as by redirecting or labeling its output, sending it signals, or specifying its reporting verbosity.

Because it performs several different roles, SRUN can be used in four distinct ways or "**modes**". These modes are described in the following table.

Mode	Description
INTERACTIVE	The simplest way to use SRUN is to distribute execution of a serial program (such as a UNIX utility) across a specified number or range of compute nodes. For example, `srun -N 8 cp ~/data1 /var/tmp/data1` copies (CP) file data1 from a common home directory to local disk space on each of eight compute nodes. SRUN allows relevant environment variables to be set on its own execute line. In interactive mode, SRUN submits job to the local SLURM job controller, then initiates all processes on the specified nodes and blocks until the requested resources become available. Many control options are available to change the details of this general pattern.
BATCH	SRUN can also directly submit complex scripts to the job queue(s) managed by SLURM for later execution, when needed resources become available and when no higher priority jobs are pending. For example, `srun -N 16 -b myscript.sh` uses the -b option of SRUN to place myscript.sh into the queue to run later on 16 nodes. Scripts in turn normally contain either MPI programs or other *simple* invocations of SRUN itself (as shown above). Thus, the -b option of SRUN supports basic, local-batch service.
ALLOCATE	The SRUN "allocate" mode can be used to combine the job complexity of scripts with the immediacy of interactive execution. For example, `srun -A -N 4 myscript.sh` uses the SRUN (uppercase) -A option to allocate specified resources (in this case, four nodes), spawn a subshell with access to those resources, and then run multiple subsequent jobs using *simple* SRUN commands within the specified script (here, myscript.sh) that the subshell immediately starts to execute.
ATTACH	To monitor or intervene in an already running SRUN job, either batch (started with -b) or interactive ("allocated", started with -A), execute SRUN again and "attach"(-a, lowercase) to that job. For example, `srun -a 6543 -j` forwards the standard output and error messages from the running job with SLURM ID 6543 to the attaching SRUN to reveal the job's current status, and (with -j, lowercase) also "joins" the job so that you can send it signals as if this SRUN had initiated the job. Omit -j for read-only attachments. Because you are attaching to a running job whose resources have already been allocated, SRUN's resource-allocation options (such as -N) are incompatible with -a.

Table 6-2. SRUN Modes

6.3.1.2 Options

For options, examples and details please refer to the man page.

Example:

```
$ man srun
```

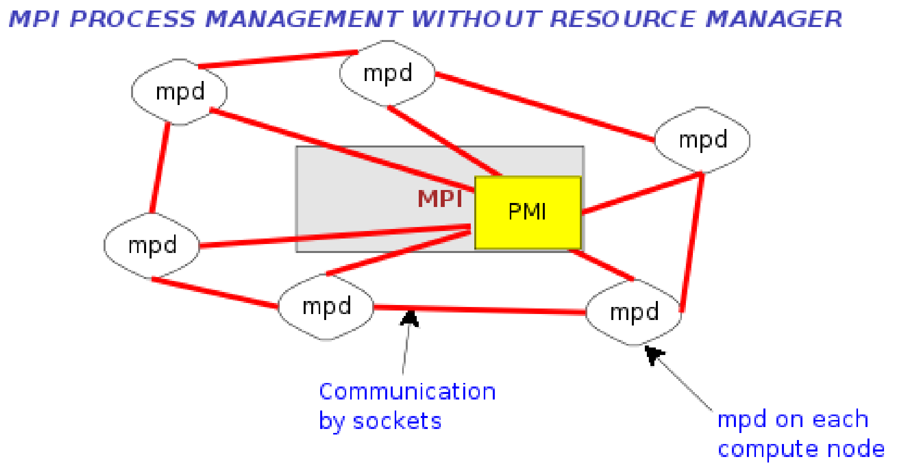
6.3.1.3 MPI Support

The **PMI** (Process Management Interface) is provided by MPIBull2 to launch processes on a cluster and provide services to the MPI interface. For example, a call to **pmi_get_appnum** returns the job id. This interface uses sockets to exchange messages.

In **MPIBull2**, this mechanism uses the mpd daemons running on each compute node. Daemons can exchange information and answer the **PMI** calls.

RMS and **SLURM** replace the Process Management Interface with their own implementation and their own daemons. No mpd is needed and when a PMI request is sent (for example pmi_get_appnum), a SLURM extension must answer this request.

The following diagrams show the difference between the use of PMI with and without a resource manager that allows process management.



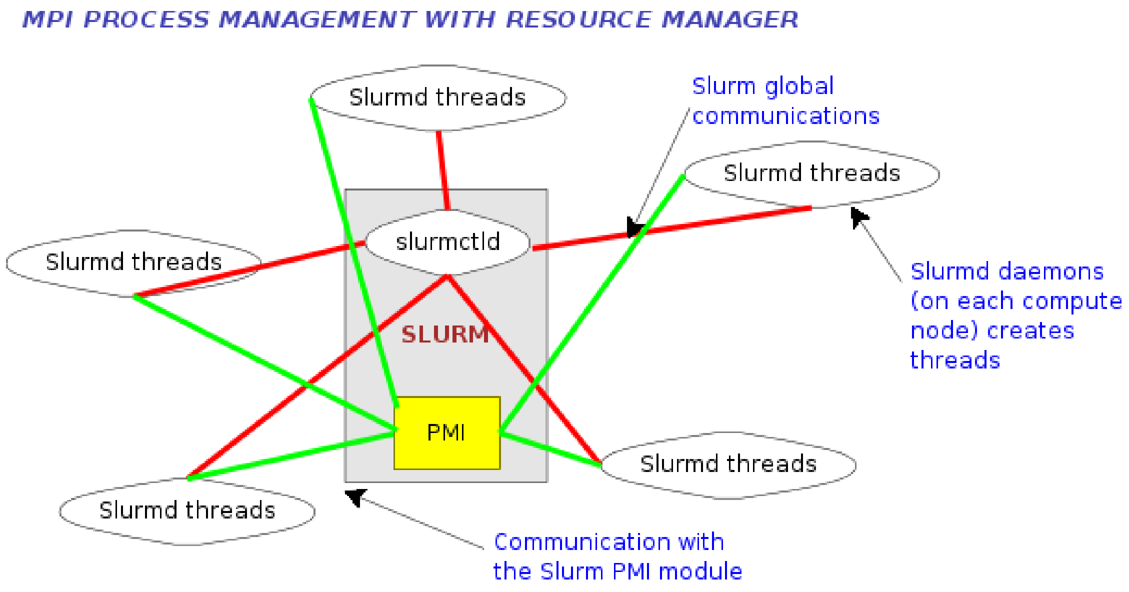

MPI PROCESS MANAGEMENT WITH RESOURCE MANAGER
Slurmd threads
Slurm global communications
Slurmd threads
slurmctld
Slurmd threads
Slurmd daemons (on each compute node) creates threads
SLURM
PMI
Slurmd threads
Slurmd threads
Communication with the Slurm PMI module

Figure 6-4. MPI Process Management With and Without Resource Manager

MPIBull2 jobs can be launched directly by the **srun** command. SLURM's *none* MPI plug-in must be used to establish communications between the launched tasks. This can be accomplished either using the SLURM configuration parameter *MpiDefault=none* in **slurm.conf** or srun's *--mpi=none* option. The program must also be linked with SLURM's implementation of the PMI library so that tasks can communicate host and port information at startup. (The system administrator can add this option to the mpicc and mpif77 commands directly, so the user will not need to bother). **Do not use SLURM's MVAPICH plug-in for MPIBull2.**

```
$ mpicc -L<path_to_slurm_lib> -lpmi ...
$ srun -n20 --mpi=none a.out
```

Notes:

- Some **MPIBull2** functions are not currently supported by the **PMI** library integrated with **SLURM**.
- Set the environment variable **PMI_DEBUG** to a numeric value of 1 or higher for the PMI library to print debugging information.

6.3.2 SBCAST

sbcast is used to copy a file to local disk on all nodes allocated to a job. This should be executed after a resource allocation has taken place and can be faster than using a single file system mounted on multiple nodes.

NAME

sbcast - transmit a file to the nodes allocated to a SLURM job.

SYNOPSIS

```
sbcast [-CfpsvV] SOURCE DEST
```

DESCRIPTION

sbcast is used to transmit a file to all nodes allocated to the **SLURM** job which is currently active. This command should only be executed within a **SLURM** batch job or within the shell spawned after the resources have been allocated to a SLURM. **SOURCE** is the name of the file on the current node. **DEST** should be the fully qualified pathname for the file copy to be created on each node. **DEST** should be on the local file system for these nodes.

Note:
Parallel file systems may provide better performance than **sbcast** can provide.

6.3.2.1 OPTIONS

For options, examples and details please refer to the man page.

Example:

```
$ man sbcast
```

6.3.3 SQUEUE (List Jobs)

SQUEUE displays (by default) the queue of running and waiting jobs (or "*job steps*"), including the JobId (used for SCANCEL), and the nodes assigned to each running job. However, SQUEUE reports can be customized to cover any of 24 different job properties, sorted by the most important properties. It also displays the job ID and job name for every job currently managed by the SLURM control daemon (SLURMCTLD). The status and resource information for each job (such as time used so far, or a list of committed nodes) are presented in a table whose content and format details can be controlled with the SQUEUE options.

NAME

SQUEUE - view information about jobs located in the SLURM scheduling queue.

SYNOPSIS

```
squeue  [OPTIONS...]
```

DESCRIPTION

SQUEUE is used to view job and job step information for jobs managed by SLURM.

6.3.3.1 OPTIONS

For options, examples and details please refer to the man page.

Example:

```
$ man squeue
```

6.3.4 SINFO (Report Partition and Node Information)

SINFO displays a summary of status information on SLURM-managed partitions and nodes (*not* jobs). Customizable SINFO reports can cover the node count, state, and name list for a whole partition, or the CPUs, memory, disk space, or current status for individual nodes as specified. These reports can assist in planning job submittals and avoiding hardware problems. The SINFO output is a table whose content and format can be controlled using the SINFO options.

NAME

SINFO - view information about SLURM nodes and partitions.

SYNOPSIS

```
sinfo [OPTIONS...]
```

DESCRIPTION

SINFO is used to view partition and node information for a system running SLURM.

6.3.4.1 OPTIONS

For options, examples and details please refer to the man page.

Example:

```
$ man sinfo
```

6.3.5 SCANCEL (Signal/Cancel Jobs)

SCANCEL cancels a running or waiting job, or sends a specified signal to all processes on all nodes associated with a job (only job owners or their administrators can cancel jobs). SCANCEL may also be used to cancel a single job step instead of the whole job.

NAME

SCANCEL - Used to signal jobs or job steps that are under the control of SLURM.

SYNOPSIS

```
scancel [OPTIONS...] [job_id[.step_id]] [job_id[.step_id]...]
```

DESCRIPTION

SCANCEL is used to signal or cancel jobs or job steps. An arbitrary number of jobs or job steps may be signaled using job specification filters or a space-separated list of specific job and/or job step IDs. A job or job step can only be signaled by the owner of that job or user root. If an attempt is made by an unauthorized user to signal a job or job step, an error message will be printed and the job will not be signaled.

6.3.5.1 Options

For options, examples and details please refer to the man page.

Example:

```
$ man scancel
```

6.3.6 SACCT (Accounting Data)

NAME

SACCT - displays accounting data for all jobs and job steps in the SLURM job accounting log.

SYNOPSIS

```
sacct options
```

DESCRIPTION

Accounting information for jobs invoked with SLURM is logged in the job accounting log file.

The **SACCT** command displays job accounting data stored in the job accounting log file in a variety of forms for your analysis. The SACCT command displays information about jobs, job steps, status, and exit codes by default. The output can be tailored with the use of the --**fields=** option to specify the fields to be shown.

For the root user, the SACCT command displays job accounting data for all users, although there are options to filter the output to report only the jobs from a specified user or group.

For the non-root user, the SACCT command limits the display of job accounting data to jobs that were launched with their own user identifier (UID) by default. Data for other users can be displayed with the --**all**, --**user**, or --**uid** options.

Note:
Much of the data reported by SACCT has been generated by the **wait3()** and **getrusage()** system calls. Some systems gather and report incomplete information for these calls; SACCT reports values of 0 for this missing data. See the **getrusage** man page for your system to obtain information about which data are actually available on your system.

6.3.6.1 Options

For options, examples and details please refer to the man page.

Example:

```
$ man sacct
```

6.3.7 Global Accounting API

Note:
The Global Accounting API only applies to clusters which use **SLURM** and the Load Sharing Facility (**LSF**) batch manager from **Platform Computing** together.

Both the **LSF** and **SLURM** products can produce an accounting file. The Global Accounting API offers the capability of merging the data from these two accounting files and presenting it as a single record to the program using this API.

Perform the following steps to call the Global Accounting API:

After SLURM has been installed (assumes **/usr** folder), build the Global Accounting API library by going to the **/usr/lib/slurm/bullacct** folder and executing the following command:

```
make –f makefile-lib
```

This will build the library **libcombine_acct.a**. This **makefile-lib** assumes that the SLURM product is installed in the **/usr** folder, and **LSF** is installed in **/app/slurm/lsf/6.2**. If this is not the case, the **SLURM_BASE** and **LSF_BASE** variables in the **makefile-lib** file must be modified to point to the correct location.

After the library is built, add the library **/usr/lib/slurm/bullacct/libcombine_acct.a** to the link option when building an application that will use this **API**.

In the user application program, add the following:

```
//  for new accounting record
//  assumes Slurm is installed under the opt/slurm folder

#include "/usr/lib/slurm/bullacct/combine_acct.h"

//  define file pointer for LSF and Slurm log file
FILE *lsb_acct_fg = NULL;    // file pointer for LSF accounting log file
FILE *slurm_acct_fg = NULL;  // file pointer for Slurm log file
int status, jobId;
struct CombineAcct newAcct;  // define variable for the new records

//  call cacct_init routine to open lsf and slurm log file,
//  and initialize the newAcct structure
status = cacct_init(&lsb_acct_fg, &slurm_acct_fg, &newAcct);

//  if the status returns 0 imply no error,
//    all log files are opened successfully.
//  then call get_combine_acct_info rountine to get the
//    combine accounting record.

//  the calling sequence is
//    int get_combine_acct_info(File *lsb_acct_fg,
//                              File *slurm_acct_fg,
//                              int   jobId,
//                              CombineAcct *newAcct);
//  where:
//  lsb_acct_fg is the pointer to the LSF accounting log file
//  slurm_acct_fg is the pointer to the Slurm accounting log file
//  jobid is the job Id from the LSF accounting log file
```

```
  //  newAcct is the address of the variable to hold the new record
  //  information.

  //  This routine will use the input LSF job ID to locate the LSF accounting
  //  information in the LSF log file, then get the SLURM_JOBID and locate the
  //  SLURM accounting information in the SLURM log file.
  //  This routine will return a zero to indicate that both records are found
  //  and processed successfully, otherwise one or both records are in error
  //  and the content in the newAcct variable is undefined.
  //  For example:

  //  to get the combine acct information for a specified jobid(2010)

    jobId = 2010;
    status = get_combine_acct_info(lsb_acct_fg,
                                   slurm_acct_fg,
                                   jobId,
                                   &newAcct);

  //  to display the record call display_combine_acct_record routine.

  display_combine_acct_record(&newAcct);

  //  when finished accessing the record, the user must close the log files and
  //  the free memory used in the newAcct variable by calling cacct_wrapup
  //  routine.
  //  For example:
  //
    if (lsb_acct_fg != NULL)               // if open successfully before
      cacct_wrapup(&lsb_acct_fg, &slurm_acct_fg, &newAcct);

  //  if an extra combine account variable is needed , the user can define
  //  the new variable and call init_cacct_rec to initialize the record
  //  and call free_cacct_ptrs to free the memory used in the new variable.
  //  For example:

  //  to define variable for the new record
      struct CombineAcct otherAcct;

  //  before using the variable otherAcct do:
     init_cacct_rec(&otherAcct);

  //  when done do the following to free the memory used by the otherAcct
  //  variable.
      free_cacct_ptrs(&otherAcct);
```

The new record contains the combined accounting information as follows:

```
/* combine LSF and SLURM acct log information */
struct CombineAcct {

         /* part one is the LSF information */

    char   evenType[50];
    char   versionNumber[50];
    time_t eventTime;
    int    jobId;
    int    userId;
    long   options;
    int    numProcessors;
    time_t submitTime;
    time_t beginTime;
    time_t termTime;
    time_t startTime;
    char   userName[MAX_LSB_NAME_LEN];
    char   queue[MAX_LSB_NAME_LEN];
    char   *resReq;
    char   *dependCond;
    char   *preExecCmd;             /* the command string to be pre_executed */
    char   fromHost[MAXHOSTNAMELEN];
    char   cwd[MAXFILENAMELEN];
    char   inFile[MAXFILENAMELEN];
    char   outFile[MAXFILENAMELEN];
    char   errFile[MAXFILENAMELEN];
    char   jobFile[MAXFILENAMELEN];
    int    numAskedHosts;
    char   **askedHosts;
    int    numExecHosts;
    char   **execHosts;
    int    jStatus;                         /* job status */
    double hostFactor;
    char   jobName[MAXLINELEN];
    char   command[MAXLINELEN];
    struct lsfRusage LSFrusage;
    char   *mailUser;                  /* user option mail string */
    char   *projectName;               /* the project name for this job, used
                                          for accounting purposes */
    int    exitStatus;                 /* job status */
    int    maxNumProcessors;
    char   *loginShell;                /* login shell specified by user */
    char   *timeEvent;
    int    idx;                        /* array idx, must be 0 in JOB_NEW */
    int    maxRMem;
    int    maxRswap;
    char   inFileSpool[MAXFILENAMELEN];  /* spool input file */
    char   commandSpool[MAXFILENAMELEN]; /* spool command file */
    char   *rsvId;
    char   *sla;             /* The service class under which the job runs. */
    int    exceptMask;
    char   *additionalInfo;
    int    exitInfo;
    char   *warningAction;               /* warning action, SIGNAL | CHKPNT |
                                            command, NULL if unspecified */
    int    warningTimePeriod;            /* warning time period in seconds,
                                            -1 if unspecified */
    char   *chargedSAAP;
    char   *licenseProject;              /* License Project */
    int    slurmJobId;                   /* job id from slurm */

     /* part two is the SLURM info minus the duplicated infomation from LSF */

    long   priority;                     /* priority */
    char   partition[64];                /* partition node */
    int    gid;                          /* group ID */
    int    blockId;                      /* Block ID */
    int    numTasks;                     /* nproc */
    double aveVsize;                     /* ave vsize */
    int    maxRss;                       /* max rss */
    int    maxRssTaskId;                 /* max rss task  */
    double aveRss;                       /* ave rss */
    int    maxPages;                     /* max pages */
    int    maxpagestaskId;               /* max pages task */
    double avePages;                     /* ave pages */
    int    minCpu;                       /* min cpu */
    int    minCpuTaskId;                 /* min cpu task */
    char   stepName[NAME_SIZE];          /* step process name */
    char   stepNodes[STEP_NODE_BUF_SIZE];  /* step node list */
    int    maxVsizeNode;                 /* max vsize node */
    int    maxRssNodeId;                 /* max rss node */
    int    maxPagesNodeId;               /* max pages node */
    int    minCpuTimeNodeId;             /* min cpu node */
    char   *account;                     /* account number */

};
```

6.4 Launching the Application using TORQUE Batch Manager

TORQUE is a resource manager providing control over batch jobs and distributed compute nodes. TORQUE uses a queue mechanism for job execution, which works according to preconfigured priority criteria.

6.4.1 Configuring Passwordless Access for TORQUE

ssh keys have to be configured to create public\private keys for an ordinary user of a cluster so that passwordless access is enabled for the whole of the cluster\partition on which the application and **TORQUE** is running. Otherwise **TORQUE** will not work correctly.

This is done by using the **ssh-keygen** command.

```
ssh-keygen -trsa
```

Append this key to the list of authorized keys.

Note:
See chapters 2 and 10 in the HPC BAS4 *Administrator's Guide* for more information on configuring **ssh**

The user command interface for TORQUE can be used to:

- Submit a job
- Display the state and characteristics of a job
- Cancel a job
- Change the characteristics of a job, which is either running or waiting. Note that for a running job, only the limits and the output files can be changed
- Stop or resume a job
- Manage more than 5000 active or waiting jobs.

For more information refer to the following Web site:
http://www.clusterresources.com/products/torque/ .

The main features of TORQUE are:

Job Priority

Users can specify the priority of their jobs.

Job-Interdependency

TORQUE enables the user to define a wide range of interdependencies between batch jobs. Such dependencies include - execution order, synchronization, and execution dependent on the success or failure of another specified job.

Automatic File Staging

TORQUE provides users with the ability to specify files that need to be copied onto the execution host before the job runs, and those that need to be copied off after the job completes. The job will be scheduled to run only after the required files have been successfully transferred.

Single or Multiple Queue Support

TORQUE can be configured with as many queues as necessary. However, TORQUE is not limited to queue-based scheduling, which means it is possible to run TORQUE with a single queue.

Multiple Scheduling Algorithms

With TORQUE it is possible to specify the standard *first-in, first-out* scheduling routine or more sophisticated algorithms.

6.4.2 TORQUE Commands

Below is a list of the most common TORQUE commands.

Command	Description
momctl	Manage/diagnose MOM (node execution) daemon
pbsdsh	Launch tasks within a parallel job
pbsnodes	View/modify batch status of compute nodes
qdel	Delete/cancel batch jobs
qhold	Hold batch jobs
qmgr	Manage policies and other batch configurations
qrls	Release batch job holds
qrun	Start a batch job
qsub	Submit jobs
qterm	Shutdown pbs server daemon

Table 6-3. TORQUE commands

qsub Command

Following is a short description of the **qsub** command. See the **qsub** man page for more details:

```
qsub - submit pbs job
```

SYNOPSIS

qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h] [-I] [-j join] [-k keep] [-l resource_list] [-m mail_options] [-M user_list]
[-N name] [-o path] [-p priority] [-q destination] [-r c] [-S path_list] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-z] [script]

DESCRIPTION

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the **-q** option is specified. See discussion of PBS_DEFAULT under Environment Variables below. Typically, the script is a shell script which will be executed by a command shell such as **sh** or **csh**.

Options for the **qsub** command allow the specification of attributes which affect the behavior of the job.

The **qsub** command will pass on certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the **qsub** command: HOME, LANG, LOGNAME, PATH, MAIL, SHELL, and TZ. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the job will have access to an environment variable named PBS_O_HOME which have the value of the variable HOME in the **qsub** command environment.

In addition to the above, the following environment variables will be available to the batch job.

PBS_O_HOST

The name of the host on which the **qsub** command is running.

PBS_O_QUEUE

The name of the original queue to which the job was submitted.

PBS_O_WORKDIR

The absolute path of the current working directory of the **qsub** command.

PBS_ENVIRONMENT

Set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job, see **-I** option.

PBS_JOBID

The job identifier assigned to the job by the batch system.

PBS_JOBNAME

The job name supplied by the user.

PBS_NODEFILE

The name of the file containing the list of nodes assigned to the job (for parallel **and cluster systems**).

PBS_QUEUE

The name of the queue from which the job is executed.

Chapter 7. Application Debugging Tools

7.1 Overview

There are two types of debuggers; symbolic ones and non-symbolic ones.

- A **symbolic debugger** gives access to a program's source code. This means that:
  - The lines of the source file can be accessed.
  - The program variables can be accessed by name.
- Whereas a **non-symbolic debugger** enables access only to the lines of the machine code program and top physical addresses.

The following tools are described in this chapter:

7.2 GDB

GDB stands for Gnu DeBugger. It is a powerful Open-source debugger, which can be used either through a command line interface, or a graphical interface such as **XXGDB** or **DDD** (Data Display Debugger). It is also possible to use an **emacs/xemacs** interface.

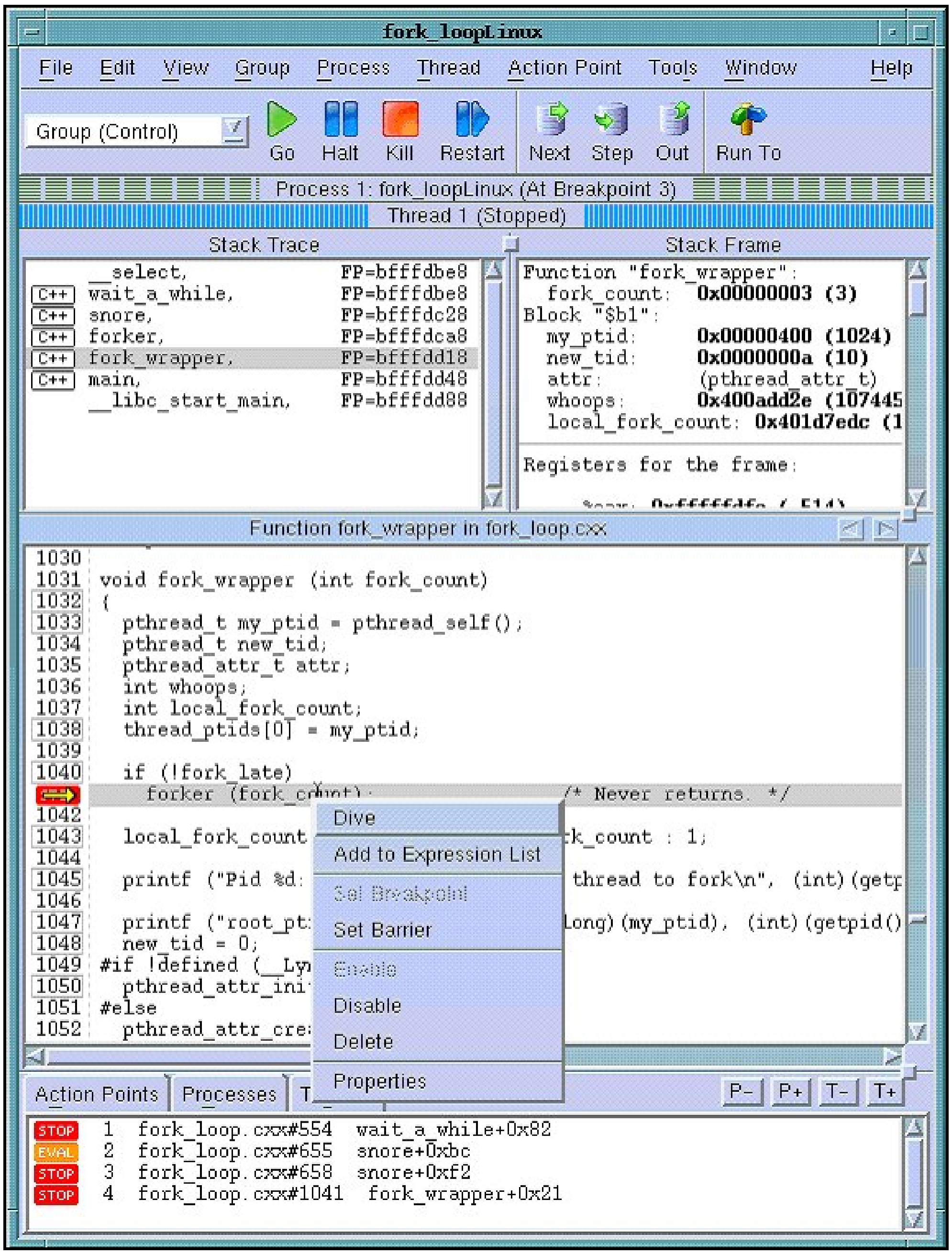
GDB supports parallel applications and threads.

GDB is published under the GNU license.

7.3 IDB

IDB is a debugger delivered with Intel compilers. It can be used with C/C++ and F90 programs.

7.4 TOTALVIEW



```
         __select,              FP=bfffdbe8
  C++    wait_a_while,          FP=bfffdbe8
  C++    snore,                 FP=bfffdc28
  C++    forker,                FP=bfffdca8
  C++    fork_wrapper,          FP=bfffdd18
  C++    main,                  FP=bfffdd48
         __libc_start_main,     FP=bfffdd88
```

```
Function "fork_wrapper":
  fork_count:    0x00000003 (3)
Block "$b1":
  my_ptid:       0x00000400 (1024)
  new_tid:       0x0000000a (10)
  attr:          (pthread_attr_t)
  whoops:        0x400add2e (107445
  local_fork_count: 0x401d7edc (1

Registers for the frame:
```

```
void fork_wrapper (int fork_count)
{
  pthread_t my_ptid = pthread_self();
  pthread_t new_tid;
  pthread_attr_t attr;
  int whoops;
  int local_fork_count;
  thread_ptids[0] = my_ptid;

  if (!fork_late)
    forker (fork_count);                        /* Never returns. */

  local_fork_count                              k_count : 1;

  printf ("Pid %d:                              thread to fork\n", (int)(getp

  printf ("root_pt                              long)(my_ptid), (int)(getpid()
  new_tid = 0;
#if !defined (__Ly
  pthread_attr_ini
#else
  pthread_attr_cre
```

```
STOP  1  fork_loop.cxx#554   wait_a_while+0x82
EVAL  2  fork_loop.cxx#655   snore+0xbc
STOP  3  fork_loop.cxx#658   snore+0xf2
STOP  4  fork_loop.cxx#1041  fork_wrapper+0x21
```

Figure 7-1 Totalview graphical interface – image taken from http://www.totalviewtech.com/productsTV.htm

TotalViewTM is a proprietary software application from Etnus and is not included with the BAS distribution. TotalviewTM is used in the same way as standard symbolic debuggers for C, C++ and Fortran (77, 90 and HPF) programs. It can also debug PVM or MPI applications. TotalViewTM has the advantage of being a debugger which supports multi-processes and multi-threading. It can take control of the various processes or threads of the program and make it possible for the user to visualize the evolution of the execution in the same window or in different windows. The processes may be local or remote.

It works just as well with mono-processor, SMP, clustered, distributed and MPP systems.

TotalView™ accepts new processes and threads exactly as generated by the application and regardless of the processor used for the execution. A process started up outside TotalView™ can also be connected to. Data tables can be filtered, displayed, and viewed in order to monitor the behavior of the program. Finally, you can descend *("call the components and details of...")* into the objects and structures of the program.

The program which needs debugging must be compiled with the option '- g', and then breakpoints should be added to the program to control its execution.

TotalView™ is an Xwindows application. Context-sensitive help provides you with basic information. You may download TotalView™ in the directory **/opt/totalview**.

Before running TotalView™, update your environment using the following command:

```
source /opt/totalview/totalview-vars.sh
```

Then enter:

```
totalview&
```

For additional information, and for copies of the documentation for Totalview™, please refer to http://www.totalviewtech.com/productsTV.htm.

7.5 MALLOC_CHECK_ - Debugging Memory Problems in C programs

When developing an application, the developer should ensure that all the buffers allocated during the run-time of the application are freed afterwards. However, even if he is vigilant, it is not unusual for memory leaks to be introduced into the code.

A simple way to detect these memory leaks is to use the environment variable **MALLOC_CHECK __**. This variable ensures that allocation routines check that each allocated buffer is freed correctly. The routines then become more 'tolerant' and allow byte overflows on both sides of blocks or for the block to be released again.

According to the value of **MALLOC_CHECK __**, when a release or allocation error appears the application behaves as follows:

- If **MALLOC_CHECK __** is set to 1, an error message is written when exiting normally.
- If **MALLOC_CHECK __** is set to 2, an error message is written when exiting normally and the process aborts. A core file is created. You should check that it is possible to create a core file by using the command *ulimit –c.* If not, enter the command *ulimit -c unlimited.*
- For any other value of **MALLOC_CHECK __**, the error is ignored and no message appears.

Example.c program:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 256

int main(void){

  char *buffer;

  buffer = (char *)calloc(256*sizeof(char));
  if(!buffer){
    perror(``malloc failed'');
    exit(-1);
  }

  strcpy(buffer, ``fills the buffer'');
  free(buffer);
  fprintf(stdout, ``Buffer freed for the first time'');
  free(buffer);
  fprintf(stdout,``Buffer freed for the second time'');
  return(0);

}
```

A program which is executed with the environmental variable MALLOC_CHECK __ set to 1 gives the following result:

```
$ export MALLOC_CHECK__=1

$./example

Buffer freed for the first time

Segmentation fault

$ ulimit -c 0

# The limit for the core file size must be changed to allow files
 bigger than 0 bytes to be generated

$ ulimit -c unlimited

# Allows an unlimited core file to be generated
```

A program which is executed with the environmental variable MALLOC_CHECK __ set to 2 gives the following result:

```
$ export MALLOC_CHECK__=2

$ ./example

Buffer freed for the first time

Segmentation fault (core dumped)
```

Example Program Analysis using the GDB Debugger

The core file should be analyzed to identify where the problem is (the program should be compiled with the option - G):

```
$ gdb example -c core
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
 and you are welcome to change it and/or distribute copies of it
 under certain conditions.
Type "show copying" to see the conditions. There is absolutely no
 warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-linux"...Using host libthread_db
 library "/lib/libthread_db.so.1".

Core was generated by `./example'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x40097354 in mallopt () from /lib/libc.so.6
(gdb) bt
#0  0x40097354 in mallopt () from /lib/libc.so.6
#1  0x4009615f in free () from /lib/libc.so.6
#2  0x0804852f in main () at exemple.c:18
(gdb)
```

The **bt** command is used to display the current memory stack. In this example the last line indicates the problem came from line 18 in the main function of the **example.c** file. Looking at the **example.c** program on page 7.4 we can see that line 18 corresponds to the second call to the free function which created the memory overflow.

7.6 Dmalloc Library

Dmalloc is an open source library and is included in the BAS distribution to help with application development and to ensure that memory leaks are detected quickly. This tool is complementary to the use of MALLOC_CHECK __ and is used to find memory leaks in C programs.

The debug memory allocation or dmalloc library is a memory management routine which provides powerful debugging facilities which are configurable at runtime. These facilities include such things as memory-leak tracking, fence-post write detection, file/line number reporting, and general logging of statistics. Thus it makes it possible to obtain precise information about a memory allocation problem. Small changes must be made to the code to run it. It also provides support for the debugging of threaded programs.

This **dmalloc** library substitutes the primitive calls **malloc**, **calloc**, **realloc** and **free** with the primitives which are available in the **dmalloc** library.

The **dmalloc** User's Guide and further information is available from the site http://www.dmalloc.com

7.7 Electric Fence

Electric Fence is an open source **malloc** debugger for Linux and Unix. It stops your program on the exact instruction that overruns or under-runs a **malloc()** buffer.

Electric Fence is installed only on the management node.

Electric Fence helps you detect two common programming bugs:

- Software that overruns the boundaries of a **malloc()** memory allocation.
- Software that touches a memory allocation that has been released by **free()**.

You can use the following example, replacing **`icc --version`** by the command line of your program.

```
[test@host ]$LD_PRELOAD=/usr/local/tools/ElectricFence-2.2.2/lib/libefence.so.0.0
icc --version

Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>

……..
```

For more details about Electric Fence please refer to http://perens.com/FreeSoftware/ .

7.8 System Monitoring and Performance Tools

In the world of HPC architectures, monitoring and improving performance is an important concern in order to fully optimize calculation speeds and memory usage for these powerful machines.

For information on monitoring tools and on improving overall performance of the application program on the HPC platform refer to the Bull HPC *Application Tuning Guide* (86 A2 19ER). This manual describes system monitoring tools provided by *NovaScale Master – HPC Edition* including **time**, **top**, and **perfmon** and application profiling tools including **gprof**, **profilecomm**, the **PAPI** library and **Intel® Trace Tools**.

For information related to the performance of the cluster itself, please refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

Appendix A. Application Troubleshooting

A list of frequently asked questions (FAQs) with solutions and advice follows:

Problems when compiling and executing

- I get the message: "`error while loading shared libraries`" when a program executes.
- My parallel program cannot find the program on the other machines.
- How do I optimize compilation with the **Intel Fortran compiler**?
- How do I optimize compilation with the **Intel C / C++ compiler**?
- Can I run applications compiled under previous OS releases?
- I get lots of "`unaligned access`" error messages.

Problems when compiling and executing with MPICH

- I have a problem with **memory allocations** when I use MPICH.
- Problems when compiling and executing with QSNET MPI.
- At runtime my program hangs when I use QSNET MPI (libelan).

OpenMP

- To run a program parallelized with OpenMP, how do I **define the number of threads** (processors) used?

I get the message: "`error while loading shared libraries`" when a program executes.

Add the path for this library to the LD_LIBRARY_PATH environment variable.

My parallel program cannot find the program on the other machines.

You must have the binaries on all machines running the benchmarks and respect the tree structure of the machine from which the benchmark is started, or use NFS.

How do I optimize compilation and debugging with the Intel Fortran compiler?

For optimization, add the following compilation options:

-implicitnone	Forces the declaration of variables: If a variable is used without being declared, this triggers errors on compilation.
-mp	Respects IEEE standard double precision.
-unroll2	To unroll a loop: This favors vectorization and the instructions pipeline.
-ip, -ipo	Optimizes calls to a subprogram (parameter management).

-auto	Allocates the variables dynamically to the stack rather than in static storage in the memory.
-zero	Implicitly initializes variables to 0.
-ftz	flush-to-zero.
-i-dynamic	Avoids loading static libraries and therefore reduces the size of the executable.
-parallel	Parallelizes certain sequences (supplied by the par_report option).
-par_report3	Provides information about how successful the compilation has been (e.g. parallelized loops).
-openmp	Takes into account OpenMP directives.

For debugging, add the following compilation options:

-g	debugging
-fpp	pre-processing

How do I optimize compilation and debugging with the Intel C / C++ compiler?

Add the following compilation options:

-O3	Highest code optimization possible.
-mp	Respects IEEE standard double precision.
-ip, -ipo	Optimizes calls to a subprogram (parameter management).
-unroll	(to unroll a loop): This favors vectorization and the instructions pipeline.

Can I run applications compiled under previous OS releases?

Some applications that have been compiled under previous OS releases (typically ISV products produced under BAS3 or RH EL AS 3) will not execute under BAS4. At runtime, the following kind of message appears:

```
symbol _dl_loaded, version GLIBC_2.2 not defined in file ld-linux-
ia64.so.2 with link time reference
```

In this case, two different and independent workarounds can be tried:

1. Set the **LD_ASSUME_KERNEL** variable to a value like 2.4, 2.4.18, or 2.4.20, and then re-start the application.

2. If the previous workaround does not solve the issue, you can create a "dummy" library that declares only the missing symbols (which is often not used):

```
echo "char* _dl_loaded=0; " > dl_loaded.c
gcc -o libdlloaded.so -shared dl_loaded.c
export LD_PRELOAD=`pwd`/libdlloaded.so
```

I get lots of “`unaligned access`” error messages.

These are not errors, but warnings. The application made an unaligned access and the processor had to get help from the kernel to access the data. This message can be ignored but be aware that too many unaligned accesses can be a source of performance loss. To hide these messages, run:

```
prctl --unaligned=silent
```

To help debugging the program, run:

```
prctl --unaligned=signal
```

I have a problem with memory allocations when I use Ethernet MPICH.

Error message displayed during execution:

```
p3_1858: (18446744073792.328125) xx_shmalloc: returning NULL; requested 65584
bytes
p3_1858: (18446744073792.328125) p4_shmalloc returning NULL; request = 65584 bytes
You can increase the amount of memory by setting the environment variable
P4_GLOBMEMSIZE (in bytes)
```

The memory that the communication requires cannot be allocated correctly. To do this, run the following command:

```
export P4_GLOBMEMSIZE=100000000
```

At runtime my program hangs when I use QSNET MPI (libelan)

If the following error message appears:

```
ELAN_EXCEPTION @ 1: 5 (Memory exhausted)
elan_createSubGroup(): Failed to allocate global Vaddr for subgroup
```

Then try again to run your program after setting the MPI_USE_LIBELAN_SUB environment variable to zero using the following command:

```
export MPI_USE_LIBELAN_SUB=0
```

To run a program parallelized with OpenMP, how do I define the number of threads (processors) used?

Run the commands:

```
export  OMP_NUM_THREADS=2 to run the program on 2 processors
export  OMP_NUM_THREADS=4 to run the program on 4 processors
```

Glossary and Acronyms

A

ANL
Argonne National Laboratory (MPICH2)

API
Application Programmer Interface

B

BAS
Bull Advanced Server

BIOS
Basic Input Output System

BMC
Baseboard Management Controller

B-SPS
Bull Scalable Port Switch

C

CLI
Command Line Interface

CMOS
Complementary Metal Oxide Semiconductor

D

DDN
DataDirect Networks S2A (storage system)

E

EFI
Extensible Firmware Interface (Intel)

EIP
IP over QSnet using Elan Kernel communications

EMP
Emergency Management Port

EPIC
Explicit Parallel Instruction set Computing

F

FAME
Flexible Architecture for Multiple Environments

FSS
FAME Scalability Switch. Each CSS Module is equipped with 2 Scalability Port Switches providing high speed bi–directional links between server components

FUTEX
Fast User mode muTEX

G

GCC
GNU C Compiler

GDB
Gnu Debugger

GNU
GNU's Not Unix

GPL
General Public License

GUI
Graphical User Interface

GUID
Globally Unique Identifier

H

HDD	Hard Disk Drive
HBA	Host Bus Adapter
HPC	High Performance Computing
HSC	Hot Swap Controller

I

ICC	Intel C Compiler
IDE	Integrated Device Electronics
IFORT	Intel Fortran Compiler
IPMI	Intelligent Platform Management Interface

K

KDM	Kernel Data Mover
KSIS	Utility for Image Building and Deployment
KVM	Keyboard Video Mouse (allows the connection of the keyboard, video and mouse either to the PAP or to the node)

L

LSF	Load Sharing Facility
LUN	Logical Unit Number

M

MDM	MPI Data Mover module
MPD	MPI Process Daemons
MPI	Message Passing Interface

N

NFS	Network File System
NPTL	Native POSIX Thread Library
NTFS	New Technology File System (Microsoft)
NUMA	Non Uniform Memory Access. A method of configuring a cluster of microprocessors in a multiprocessing system so that they can share memory locally, improving performance and the ability of the system to be expanded.
NVRAM	Non Volatile Random Access Memory

O

OEM	Original Equipment Manufacturer
OPK	OEM Preinstall Kit (Microsoft)

P

PAM	Platform Administration and Maintenance software
PAP	Platform Administration Processor
PAPI	Performance Application Programming Interface
PCI	Peripheral Component Interconnect (Intel)
PDU	Power Distribution Unit
PM	Process Manager
PMB	Platform Management Board
PMI	Process Management Interface
PMU	Performance Monitoring Unit
PRUN	Parallel Run (Quadrics)
PVFS	Parallel Virtual File System
PVM	Parallel Virtual Machine

Q

QBB	Quad Brick Board. The QBB is the heart of the Bull NovaScale 5xxx/6xxx Series platforms, housing 4 Itanium _ 2 processors.

R

RMS	Resource Management Service (Quadrics)
RPM	RPM Package Manager

S

SCI	Scalable Coherent Interconnect
SDR	Sensor Data Record
SDP	Sockets Direct Protocol
SEL	System Event Log
SCSI	Small Computer System Interface
SLURM	Simple Linux Utility for Resource Management
SM	System Management
SMP	Symmetric Multi Processing. The processing of programs by multiple processors that share a common operating system and memory.
SNMP	The protocol governing network management and the monitoring of network devices and their functions.
SOL	Serial Over LAN
SSH	Secure Shell

U

UA

User’s Application

V

VGA

Video Graphic Adapter

Index

Technical publication remarks form

Title:	HPC BAS4 User's Guide

Reference:	86 A2 29ER 09	Date:	April 2008

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME: ______________________ DATE: __________
COMPANY: ______________________
ADDRESS: ______________________

____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone:	+33 (0) 2 41 73 72 66
FAX:	+33 (0) 2 41 73 70 66
E-Mail:	srv.Duplicopy@bull.net

Reference	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ______________________________ DATE: ____________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ______________________ FAX: ______________

E-MAIL: ______________________________

For Bull Subsidiaries:
Identification: ______________________________

For Bull Affiliated Customers:
Customer Code: ______________________________

For Bull Internal Customers:
Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
86 A2 29ER 09